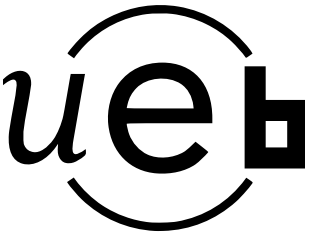
Caractérisation locale de fautes dans les systèmes large échelle

Romaric Ludinard

ANNÉE 2014



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Romaric LUDINARD

préparée à l'unité de recherche INRIA Rennes - Bretagne Atlantique
Université de Rennes 1

Caractérisation locale de fautes dans les systèmes large échelle

Thèse soutenue à Rennes
le 02 octobre 2014

devant le jury composé de :

Ludovic MÉ / *Président*
Professeur, Supélec, Rennes

Roberto BALDONI / *Rapporteur*
Full Professor, Sapienza Università Di Roma, Rome

Sébastien TIXEUIL / *Rapporteur*
Professeur, UPMC, LIP6, Paris

Emmanuelle ANCEAUME / *Co-encadrante de thèse*
Chargée de recherche CNRS, IRISA, Rennes

Marc-Olivier KILLIJIAN / *Examinateur*
Directeur de recherche CNRS, LAAS, Toulouse

Erwan LE MERRER / *Examinateur*
Chercheur, Technicolor R&I, Rennes

Bruno SERICOLA / *Directeur de thèse*
Directeur de Recherche, Inria, Rennes

François TAIANI / *Examinateur*
Professeur, Université de Rennes 1, Rennes

Remerciements

Les travaux présentés dans ce manuscrit n'auraient jamais pu voir le jour sans le concours et le soutien d'un certain nombre de gens, tant durant la thèse elle-même que pendant la (longue) période de cheminement qui l'a précédée. Malheureusement, la mise en place d'une liste exhaustive de remerciements est une tâche ardue et je vais certainement en oublier certains. J'espère ne pas commettre trop de fautes et que les personnes concernées ne m'en tiendront pas rigueur. Afin de simplifier cet exercice et tenter de limiter mes propres oublis, je vais procéder à des remerciements par ensemble cohérent de personnes et de manière à peu près antéchronologique.

Pour commencer, je souhaite remercier Ludovic Mé, Professeur à Supélec, pour avoir accepté de présider mon jury de doctorat, ainsi que pour les diverses et riches interactions que nous avons pu avoir au cours des différents projets menés avant ma thèse. Je remercie très chaleureusement Roberto Baldoni, Full Professor à l'Université La Sapienza à Rome, et Sébastien Tixeuil, Professeur à l'UMPC / LIP6, pour leurs lectures que j'imagine nombreuses et attentives, leurs remarques et critiques vis-à-vis de ce manuscrit. Enfin, je souhaite remercier Marc-Olivier Killijian, Directeur de Recherche CNRS au LAAS, et François Taiani, Professeur à l'Université de Rennes 1, d'avoir accepté le rôle d'examinateur dans ce jury.

Mes remerciements suivants s'adressent aux personnes avec qui j'ai pu travailler directement pendant ces trois ans (et quelques mois) qui sont les piliers de ces travaux. Il est évident que sans leur concours, cette thèse serait tout autre.

Un grand merci à Erwan Le Merrer et Gilles Straub de Technicolor pour leur travail en amont de cette thèse. Merci aussi pour l'accueil au sein de l'équipe, l'accompagnement, les discussions, le travail mené et la souplesse dont ils ont parfois du faire preuve. Un grand merci aussi à Erwan pour l'interface avec Willem et Pierre, et à Gilles pour l'interface avec d'autres.

Je remercie bien évidemment Bruno Sericola, mon directeur de thèse, pour son encadrement, son soucis du détail et de la précision ainsi que pour ses nombreuses remarques pertinentes. Enfin, une reconnaissance infinie à Emmanuelle Anceaume, non pas tant pour l'encadrement pendant ces trois années de doctorat que pour la totalité du travail accompli ensemble depuis 8 ans. Merci pour son soutien, sa compréhension et son enthousiasme communicatif. Mes excuses au passage à nos voisins de bureau qui nous ont incité plusieurs fois à fermer la porte du bureau...

J'ai le plaisir d'être Attaché Temporaire d'Enseignement et de Recherche à l'ISTIC pour l'année universitaire 2014-2015. Parmi les personnes qui constituent les

différentes équipes pédagogiques auxquelles j'appartiens, je souhaite remercier tout particulièrement Delphine Demange et Catherine Belleannée pour leur compréhension, leur support et leur soutien dans les semaines qui ont précédé ma soutenance de thèse.

Je tiens tout particulièrement à remercier Yann Busnel et Jean-Louis Marchand que je connais depuis un certain temps, mais avec qui j'ai eu l'occasion de collaborer pendant ma thèse. Cette collaboration a mené à l'un des résultats centraux de ce manuscrit et de mes travaux de thèse.

Pendant la durée de ma thèse j'ai été bilocalisé entre mon bureau au centre de recherche Inria Rennes - Bretagne Atlantique au sein de l"équipe Dionysos et Technicolor R&I à Rennes. Je remercie chaleureusement les membres de ces deux équipes pour leur accueil et les discussions, séminaires, cafés, sprints... que nous avons pu partager. Un clin d'œil particulier à Alexandre Van Kempen et Nicolas Le Scouarnec pour le travail mené sur le papier Gateways.

Avant de commencer cette thèse je suis intervenu dans différents projets de recherche en tant qu'ingénieur. Ces expériences se sont montrées très riches et font partie du chemin qui m'a mené à cette thèse. Je remercie les personnes avec qui j'ai pu travailler dans le cadre de ces projets, et plus particulièrement Frédéric Tronel, Michel Hurfin, Jean-Pierre Le Narzul, Éric Totel, Frédéric Majorczyk, Aina Ravoaja, Francisco Brasileiro et Erwan Abgrall.

Cette particularité a fait les joies d'un certain nombre de gens qui ont du travailler à mettre en place les modalités pour que le financement de cette thèse soit possible. Je remercie tout spécialement Caroline Lebaron et Thierry Gélin qui étaient en première ligne pour la mise en place des aspects légaux de cette thèse.

Je dois enfin remercier les gens de mon entourage. Je remercie tout d'abord ma famille, entre autres pour le support question vignes quand j'étais pris par la rédaction et encore une fois ma mère, en particulier pour le support culinaire pour le pot de thèse.

Un grand merci à Sara pour sa présence et son soutien, en particulier pour les moments où la recherche déborde sur nos moments à nous. Merci encore à elle d'avoir chapeauté une bonne part du off du pot de thèse de manière transparente pour moi, et merci d'avance pour ce qu'il reste à venir.

Merci à ceux qui ont suivi la même voie et qui ne sont certainement pas étrangers à ce cheminement : la joyeuse compagnie de la smash-bavette (Jérémy, Benoit Gronchon, JPio, Fanfoué), les (ex-)doctorants Kostas, Antoine, Héverson, Erwan Raffin, Julien Stainer...

Merci aux amis, ceux présents pour mes 30 ans à la conférence OPODIS 2012 : Yves-Alexis, Marie, Delf, Alex, Fred, merci aussi à Pierre-Louis, Julien & Solenn, à ceux qui n'ont pu être là : Ben Roux & Solène, Jean-Loup, Nono... et enfin aux adeptes des chaînes de mails infinies : Marie & Alex, Marine & Vincent, Jé, Thom, Mathieu, Élo.

Table des matières

Introduction

Internet est un réseau de réseaux sans centre névralgique servant de support à divers services tels que les communications électroniques, la messagerie instantanée, la vidéo à la demande ou le World Wide Web. L'accès à Internet et son utilisation au quotidien se sont largement démocratisés depuis sa création. En France, l'autorité de régulation des communications électroniques et des postes [ARC13] recensait au troisième trimestre 2013, environ 24 millions d'abonnements au haut et très haut débit. Ces abonnés sont répartis entre quelques fournisseurs d'accès à Internet. Un fournisseur d'accès à Internet, ou opérateur de réseau, gère un réseau qui lui est propre et qui est connecté à d'autres réseaux au sein d'Internet. Un fournisseur d'accès à Internet fournit à ses utilisateurs une connexion à Internet. Chaque utilisateur se connecte à Internet au travers d'un équipement appelé "passerelle de connexion". Ainsi connecté, l'utilisateur peut ainsi utiliser les services basés sur Internet.

Exemple 1 *La figure 1 représente une interconnexion possible de trois réseaux de fournisseurs d'accès à Internet (*Fai 1*,* Fai 2 *et* Fai 3*). Chaque utilisateur est connecté à Internet au travers d'une passerelle de connexion Internet représenté par les ronds bleus. Les passerelles de connexion sont reliées à des routeurs R_i. Les routeurs sont reliées entre eux.*

Les fournisseurs de services déploient des serveurs au sein d'Internet afin de délivrer un service aux utilisateurs. La figure 1 illustre un positionnement possible de ces serveurs. La notation S_x^y représente le serveur x délivrant le service s_y. Par exemple, le service s_1 est délivré par les serveurs S_1^1, S_5^1 et S_6^1. Notons que le service s_1 est délivré par un serveur présent dans chacun des trois réseaux. À l'inverse le service s_4 n'est délivré que par un unique serveur S_4^4 présent dans le réseau géré par Fai 2*. Enfin, les utilisateurs* 1 *à* 6 *peuvent accéder au service s_2 par le serveur S_8^2.*

Supervision

Malheureusement, chacun des éléments présents dans le réseau ou impliqués dans les services consommés par les utilisateurs peut potentiellement exhiber des défaillances. Par exemple, un routeur peut omettre de renvoyer temporairement le trafic qu'il reçoit, sa bande passante peut être saturée ou il peut dupliquer des messages. De la même manière, des messages peuvent être perdus sur les liens de communication, les serveurs

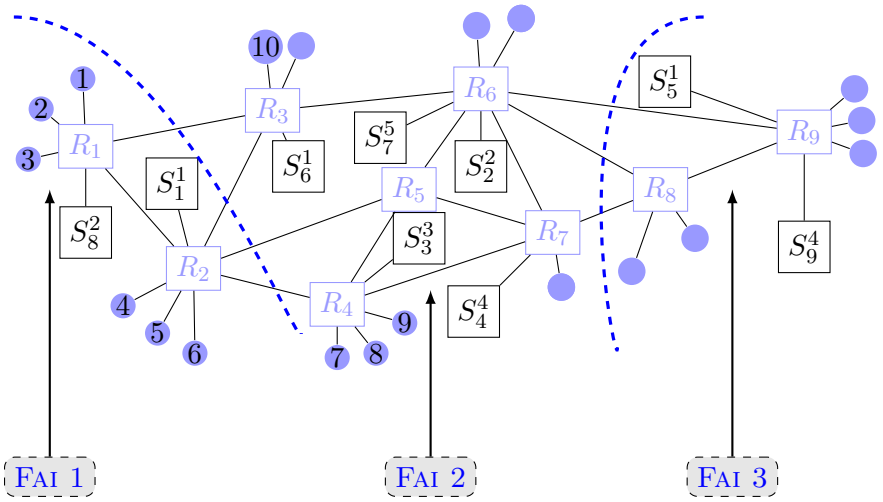


FIGURE 1 – Disposition de serveurs au sein d'Internet

peuvent être temporairement indisponibles et les passerelles de connexions peuvent avoir un comportement dégradé voire devenir inopérantes.

La supervision d'un système est la tâche qui consiste à collecter et analyser des données du système de manière continue afin de s'assurer du bon fonctionnement de celui-ci. La supervision d'un système permet ainsi de déceler si le système devient défaillant, c'est-à-dire que le service qu'il délivre dévie du service attendu. Les fournisseurs d'accès à Internet et de services supervisent le système afin d'intervenir rapidement lorsqu'une défaillance est détectée.

Les protocoles de supervision tels que Simple Network Managment Protocol (SNMP) [CFSD90] ou TR-69 [Bro] définissent le comportement des entités supervisées ainsi que leurs interactions avec le superviseur qui collecte et analyse ces données. Les entités supervisées exécutent une supervision locale de leur état interne. Ces données sont disponibles à la demande du superviseur ou lui sont envoyées périodiquement. En cas de défaillance, un administrateur peut exécuter des procédures locales sur l'entité supervisée à des fins de diagnostic. Le diagnostic est la tâche d'identification des causes d'une défaillance.

Ces protocoles sont largement employés dans le cadre de la supervision d'éléments réseaux comme les routeurs. Le service assuré par ces éléments est local et consiste à faire transiter des informations d'une interface vers une autre en fonction d'un ensemble de règles préétablies appelée "politique de routage". Par exemple, un routeur se montre défaillant s'il ne fait plus transiter les informations, les fait transiter de manière erronée ou s'il met trop de temps à le faire. D'autre part, les protocoles de routage réseau sont redondants. De cette manière, si un routeur est défaillant, le service est délivré en empruntant un chemin alternatif. La défaillance du routeur est ainsi masquée à l'utilisateur et permet à l'opérateur de rechercher les causes de cette défaillance, sans rupture de service.

Exemple 2 *Revenons à l'exemple décrit dans la figure 1 et considérons que l'utilisateur* 4 *consomme le service* s_4*. Celui-ci est délivré par* S_4^4 *en empruntant les routeurs* R_2*,* R_5 *et* R_7*. Si le routeur* R_5 *devient défaillant, un chemin alternatif est emprunté de manière transparente pour l'utilisateur, par exemple* R_2*,* R_4 *et* R_7*.*

Problématique considérée

À l'inverse des équipements réseaux qui assurent un service local et dont on peut masquer les défaillances par l'utilisation de chemins redondants, les passerelles de connexion constituent un point unique de défaillance. En effet, une défaillance de la passerelle de connexion amène à une dégradation des services consommés par l'utilisateur. Par conséquent, il est primordial pour l'opérateur de superviser ces équipements afin de diagnostiquer au plus vite une défaillance et assurer la connexion de l'utilisateur à Internet.

De manière similaire aux équipements réseau, on peut imaginer que chaque passerelle de connexion supervise la qualité des services consommés. Ainsi, lorsque la passerelle de connexion est défaillante, celle-ci peut notifier l'opérateur réseau. Dans la pratique cette approche n'est pas employée. L'exemple suivant illustre les limites de cette approche.

Exemple 3 *Revenons à l'exemple décrit dans la figure 1 et considérons que l'utilisateur* 5 *consomme le service* s_2*.*

Cas 1 *Supposons que la passerelle de connexion* 5 *soit défaillante, la qualité du service* s_2 *perçue par la passerelle diminue, la passerelle envoie une alarme à l'opérateur.*

Cas 2 *Supposons à présent que le service* s_2 *est délivré à l'utilisateur* 5 *par le serveur* S_8^2*. Considérons de plus que le serveur* S_8^2 *est défaillant, le service* s_2 *est délivré par le serveur* S_2^2*. Dans ce cas, l'utilisateur* 5 *perçoit une variation de la qualité du service* s_2 *liée au changement de serveur envoie une alarme à l'opérateur. Pourtant, la passerelle de connexion* 5 *n'est pas défaillante, l'alarme est inutile (faux positif).*

Cas 3 *Considérons à présent que les utilisateurs* $1, 2, 3, 4, 5$ *et* 6 *consomment le service* s_2 *délivré par le serveur* S_8^2*. De la même manière, si le serveur* S_8^2 *est défaillant, le service* s_2 *pourra être délivré par le serveur* S_2^2*. Dans ce cas, chaque utilisateur perçoit une variation de qualité du service* s_2 *et envoie donc une alarme à l'opérateur. Pourtant, aucune des passerelles de connexion* $1, 2, 3, 4, 5, 6$ *n'est défaillante, ces alarmes sont inutiles.*

Afin d'éviter de surcharger l'opérateur avec des alarmes non pertinentes, ce type d'approche est désactivé dans la pratique.

Comme l'illustre cet exemple, cette approche peut engendrer de nombreux faux positifs. Pour cette raison, ce type d'approche pour les passerelles de connexion est désactivé dans la pratique. Les protocoles SNMP ou TR-69 ne sont utilisés que pour des tâches d'administration ou de diagnostic. Les fournisseurs d'accès à Internet pallient

l'absence de détection de défaillance automatique en déléguant cette tâche aux utilisateurs finaux et en utilisant des centres d'appels. Un utilisateur peut ainsi contacter un centre d'appel s'il perçoit une défaillance et un technicien peut éventuellement rechercher les causes de la défaillance perçue par l'utilisateur. Bien qu'utilisée en pratique, cette approche n'est pourtant pas satisfaisante pour trois raisons. Tout d'abord, cette solution nécessite une activité humaine permanente pour répondre aux sollicitations des utilisateurs et éventuellement chercher à localiser la défaillance perçue ainsi que les causes de celle-ci. D'autre part, il peut y avoir un délai important entre l'apparition de la défaillance et la notification de l'utilisateur, rendant ainsi plus difficile le travail de diagnostic. Enfin, les utilisateurs peuvent contacter le centre d'appel de l'opérateur pour des raisons indépendantes du fonctionnement de la passerelle de connexion, comme illustré dans les cas 2 et 3 de l'exemple 3, nuisant ainsi à l'efficacité d'une telle solution.

Exemple 4 *Revenons à l'exemple décrit dans la figure 1 et considérons le cas 3 de l'exemple précédent. Dans ce cas, les utilisateurs* $1, 2, 3, 4, 5$ *et* 6 *consomment le service* s_2 *délivré par le serveur* S_8^2*. Si le serveur* S_8^2 *est défaillant, le service* s_2 *pourra être délivré par le serveur* S_2^2*. Dans ce cas, chaque utilisateur perçoit une variation de qualité du service* s_2 *et envoie donc une alarme à l'opérateur. Ces alarmes ne sont pas révélatrices de défaillances des passerelles de connexion* $1, 2, 3, 4, 5, 6$ *et sont donc considérées comme des faux positifs. Afin d'éviter la trop forte proportion de faux positifs, ce type d'alarmes est désactivé dans la pratique.*

En effet, l'approche actuelle considère que chacune de ces alarmes, prise individuellement, constitue un faux positif, et donc que celles-ci n'apportent aucune information. Pour autant, si on considère l'ensemble de ces alarmes, elles sont révélatrices d'une défaillance perçue par un grand nombre d'utilisateurs. Il semble donc intéressant d'utiliser ces alarmes afin de faire la distinction entre les défaillances perçues par un petit nombre d'utilisateurs et les défaillances perçues par un grand nombre d'utilisateurs.

L'exemple 4 nous montre que les variations de qualités perçues par les utilisateurs sont corrélées lorsqu'ils perçoivent une défaillance due à une même cause appelée faute. Il semble donc intéressant d'exploiter cette corrélation afin de distinguer d'une part les défaillances perçues par un petit nombre d'utilisateurs, comme dans le cas d'une défaillance de passerelle de connexion, de celles perçues par un grand nombre d'utilisateurs, comme dans le cas d'une défaillance d'un équipement réseau ou d'un serveur. Dans le premier cas, nous parlerons de "faute isolée" tandis que dans le second cas nous parlerons de "faute massive". La question qui se pose naturellement est alors la suivante :

Dans quelle mesure et sous quelles conditions est-il possible de distinguer les fautes isolées des fautes massives, en ne se basant que sur les perceptions des utilisateurs ?

Contexte de cette étude

L'exemple 4 illustre l'intuition que nous allons développer dans ce document : les utilisateurs ayant perçu des variations de qualités similaires sont susceptibles d'avoir perçu une même défaillance. Cependant, afin de clarifier cette intuition, l'exemple 4 occulte certains aspects du contexte considéré. Tout d'abord, cette étude se place dans le contexte d'un réseau de fournisseur d'accès à Internet. Par conséquent, nous considérons ici plusieurs milliers (voire millions) d'utilisateurs connectés et consommant divers services. D'autre part, nous considérons que plusieurs fautes sont susceptibles d'impacter le système dans un intervalle de temps restreint. Dans ce cas, le nombre d'alarmes dues à des variations de qualités peut être très grand et toutes les variations de qualités perçues ne sont donc pas nécessairement corrélées entre elles. Enfin, la figure 1 décrit représente une interconnexion possible de trois réseaux de fournisseurs d'accès à Internet ainsi que le placement des serveurs et des utilisateurs dans cette interconnexion. Cette illustration est utile pour décrire la situation de l'exemple 4 mais n'est pas utilisée pour regrouper les variations de qualités similaires. Par conséquent, les travaux décrits dans ce document ne s'appuient pas sur la connaissance du réseau d'interconnexion, la position des différents serveurs et utilisateurs dans le réseau.

Organisation du document

Le reste de ce document est organisé en huit chapitres.

Chapitre 1 Ce chapitre décrit les concepts nécessaires aux travaux décrits dans ce document. Plus précisément, les notions de faute, erreur et défaillance, relatives à la sûreté de fonctionnement sont rappelées. Dans un second temps, la notion de système large échelle est présentée.

Chapitre 2 Ce chapitre donne un aperçu des principales approches employées dans la conception de systèmes large échelle. Ensuite, ce chapitre décrit différentes familles d'algorithmes de clustering de données, le clustering étant la tâche consistant à regrouper les données similaires ensemble. Enfin, les principaux travaux existants sur la supervision et la caractérisation de fautes dans les systèmes large échelle sont présentés et discutés.

Chapitre 3 Ce chapitre décrit un cadre idéal permettant de modéliser l'impact des fautes dans le système sur les qualités des services perçues par les entités supervisées. Malgré ce cadre idéal, nous montrons qu'il est impossible de déterminer de manière certaine, pour chaque entité supervisée, si la défaillance qu'elle perçoit est due à une faute isolée ou à une faute massive. Cependant, en assouplissant les choix possibles pour la caractérisation du type de faute (*isolée*, *massive* ou *indéterminée*), on peut alors déterminer pour chaque variation anormale de qualité, le type de faute à lui associer. Cette caractérisation ne nécessite la connaissance que d'un sous-ensemble des entités

du système. Celle-ci s'appuie sur la connaissance des entités percevant des qualités et des variations similaires et est aussi précise que celle que ferait un observateur ayant accès à l'ensemble des entités du système.

Chapitre 4 Ce chapitre fournit les algorithmes nécessaires à la caractérisation des fautes décrite au chapitre 3. Nous montrons que ces algorithmes peuvent être exécutés localement et ne requièrent qu'une connaissance partielle des entités du système. Ces algorithmes sont évalués et comparés au travers de diverses simulations. Enfin, nous discutons de la pertinence du modèle que nous avons mis en place.

Chapitre 5 Ce chapitre décrit la mise en place de FixMe, une architecture auto-organisée permettant aux entités de trouver de manière efficace les autres entités percevant des qualités similaires et ayant subi les mêmes variations de qualité. Les algorithmes décrivant les différentes opérations à la mise en œuvre de cette architecture sont décrits, ainsi que la complexité algorithmique de ces opérations. Enfin, la mise en œuvre des algorithmes décrits dans le chapitre 4 dans le cadre de cette architecture est décrite.

Chapitre 6 FixMe est formé de deux niveaux, : le premier permettant gérer les entités suivant la qualité perçue et le second, nommé PeerCube, gérant les entités percevant une même qualité. Ce chapitre présente le fonctionnement basique de PeerCube puis une évaluation de performance de ces composants élémentaires. Nous y étudions le comportement de ces composants dans le pire cas d'exécution, c'est-à-dire en présence d'un adversaire visant à corrompre le système.

Chapitre 7 Ce chapitre présente une évaluation de performance de PeerCube dans son ensemble dans le pire cas d'exécution. En s'appuyant sur l'étude menée au chapitre 6, nous montrons que cette architecture est résiliente au churn (*i.e.*, les entrées / sorties du système ont un impact restreint sur celui-ci) et qu'un adversaire n'a qu'un pouvoir limité sur cette architecture.

Chapitre 8 Ce chapitre termine le présent document. Les différents chapitres et contributions y sont résumés. Les pistes d'améliorations ainsi que les questions ouvertes et pistes de réflexion sont ensuite présentées.

Contributions et publications

Les publications relatives au présent document apparaissent dans la section bibliographie de ce chapitre d'introduction. Plus spécifiquement, les contributions dont je suis le principal contributeur portent sur :

— Les concepts, la modélisation et la mise en œuvre algorithmique abordés aux chapitres 3 et 4. Ces travaux ont été présenté à la conférence DSN [ABL+14a]

et à AlgoTel [ABL+14b]. De plus, ces travaux ont conduit à un dépôt de brevet [LLSS13] en collaboration avec Technicolor.
— L'architecture présentée dans le chapitre 5. Ce travail a été présenté à OPODIS en 2012 [ALL+12] puis à AlgoTel en 2013 [ALL+13b] ainsi que dans une conférence industrielle [ALL+13a]. De plus, ce travail a fait l'objet d'un dépôt de brevet [LLSS12] en collaboration avec Technicolor.
— La modélisation probabiliste de PeerCube abordée au chapitre 6. Ce travail a été présenté en 2011 à DSN [ASLT11] et à CFIP [ALST11].

La liste complète de mes publications se situe à la fin du présent document.

Bibliographie

[ABL+14a] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola : Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, juin 2014.

[ABL+14b] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand, B. Sericola et G. Straub : Anomaly Characterization Problems. Dans *16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, 2014.

[ALL+12] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : A Self-organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. Dans *Proceedings of the 16th International Conference On Principles Of Distributed Systems*, OPODIS, pages 1–12, décembre 2012.

[ALL+13a] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : A Self-organising Isolated Anomaly Detection Architecture for Large Scale Systems. Dans *Nem-Summit*, octobre 2013.

[ALL+13b] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : Détection Répartie de Défaillances Isolées. Dans *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, mai 2013.

[ALST11] E. Anceaume, R. Ludinard, B. Sericola et F. Tronel : Modélisation et Évaluation des Attaques Ciblées dans un Overlay Structuré. Dans *Colloque Francophone sur l'Ingénierie des Protocoles*, CFIP, 2011.

[ARC13] ARCEP : Haut et très haut débit sur réseaux fixes au 30 septembre 2013. `http://www.arcep.fr/index.php?id=12115`, novembre 2013.

[ASLT11] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel : Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, juin 2011.

[Bro] Broadband Forum : TR-069 CPE WAN Management Protocol Issue 1, Amend.4, 2011.

[CFSD90] J. D. Case, M. Fedor, M. L. Schoffstall et J. Davin : Simple Network Management Protocol (SNMP). Rapport technique, IETF, 1990.

[LLSS12] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : Method for Isolated Anomaly Detection in Large-scale Data Processing Systems. patent no. 12306237.4, octobre 2012.

[LLSS13] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : Method for Isolated Anomaly Detection in Large-scale Audio/Video/Data Processing Systems. patent no. 13306029.3, juillet 2013.

Chapitre 1

Terminologie et modèle

Ce chapitre rappelle les définitions que nous utiliserons tout au long de ce document et présente le contexte dans lequel se place nos travaux, à savoir les systèmes large échelle.

1.1 Terminologie

Nous rappelons dans cette section les définitions de Jean-Claude Laprie employées dans [Lap96, ALRL04].

Fautes, erreurs et défaillances Un *système* est une entité qui interagit avec d'autres entités afin de rendre un *service*. L'utilisateur est une entité particulière qui interagit avec le système considéré afin d'utiliser le service qu'il délivre. Le service est correct si le service délivré accomplit la fonction du système, *i.e.*, ce pour quoi le système est destiné. La sûreté de fonctionnement d'un système est définie comme "la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans la qualité du service qu'il leur délivre".

Un système est *défaillant* lorsque le service qu'il délivre dévie du service correct ou attendu. Une défaillance est la conséquence d'une *faute* dans le système. Un état d'erreur désigne l'état anormal du système, résultant de l'activation d'une faute et pouvant potentiellement amener à une défaillance. L'erreur permet de lier la faute à la défaillance perçue par l'utilisateur.

Ces notions sont dépendantes du système considéré. Ainsi, un même événement pourra être une faute, une erreur ou une défaillance suivant le point de vue fonctionnel considéré. Considérons un système constitué d'un serveur délivrant un service. En cas de panne électrique (faute), le serveur n'est plus alimenté (erreur) et ne peut plus délivrer son service (défaillance).

Un système est généralement composé de sous-systèmes qui interagissent entre eux. Une défaillance de l'un de ces sous-systèmes peut-être considérée comme une faute dans le système global.

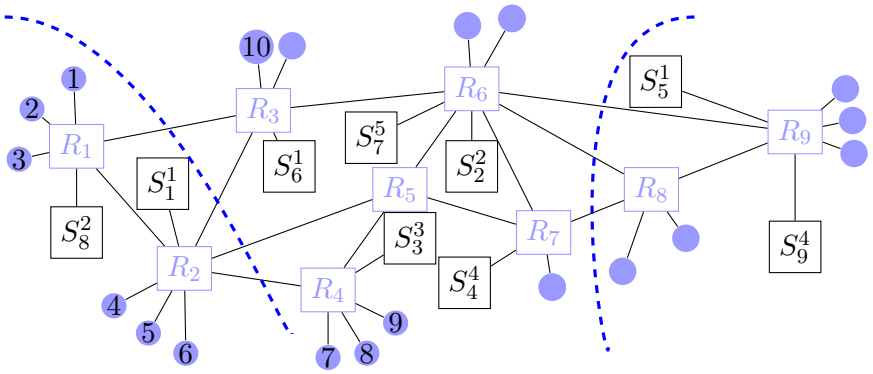


FIGURE 1.1 – Disposition de serveurs au sein d'Internet

Modèles de défaillances L'activation d'une faute dans le système peut avoir des conséquences variables. Les défaillances peuvent être catégorisées suivant deux aspects : le domaine de défaillance et la cohérence des défaillances.

Le premier aspect porte sur le service rendu par le système et le délai nécessaire pour fournir ce service. On parle de défaillance en valeur lorsque le système délivre une valeur ou un service erroné. Lorsque le service n'est pas délivré au moment attendu (trop tôt ou trop tard), on parle de défaillance temporelle. Lorsque le service n'est plus rendu, on parle de défaillance par arrêt.

D'autre part, on dit d'une défaillance qu'elle est cohérente si tous les utilisateurs perçoivent la même déviation du service vis-à-vis du service attendu. Enfin, si le comportement d'une entité du système dévie de manière arbitraire de son comportement normal ou attendu, on parle de défaillance byzantine. Dans un système de communication par message, l'absence d'émission de message par une entité peut constituer une défaillance cohérente. À l'inverse, une entité envoyant des messages avec un contenu incorrect (défaillance en valeur), au mauvais moment (défaillance temporelle) ou non prévus constitue une défaillance byzantine.

Exemple 5 *La figure 1.1 illustre un exemple de système que nous considérons dans le cadre de ce document. Le système est constitué d'une ensemble de routeurs interconnectés par des liens, de serveurs délivrant des services et de passerelles utilisateurs. Chaque utilisateur consomme des services proposés par le système de manière transparente comme illustré sur la figure 1.2.*

Dans cette figure, le service s_2 est délivré par les serveurs S_2^2 et S_8^2, tandis que le service s_3 n'est fourni que par le serveur S_3^3. L'utilisateur 5 consomme ces deux services s_2 et s_3.

Considérons le serveur S_3^3 comme un sous-système du système considéré. Si une panne électrique (faute) se produit au niveau du serveur S_3^3, celui-ci n'est plus alimenté (erreur), le service s_3 n'est plus rendu (défaillance). Dans le contexte du système global, la défaillance du serveur S_3^3 constitue une faute. Le serveur S_3^3 est le seul à délivrer

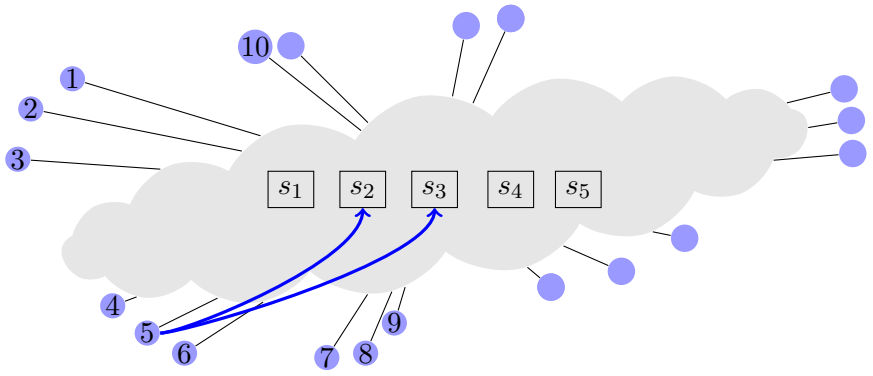


FIGURE 1.2 – Accès au services

le service s_3. Celui-ci n'est plus rendu, il s'agit d'une défaillance par arrêt. Tous les utilisateurs du système souhaitant consommer le service s_3 perçoivent cette défaillance. Il s'agit d'une défaillance cohérente.

Considérons à présent le serveur S_8^2 comme un sous-système du système considéré. Dans les mêmes circonstances, si une panne électrique (faute) se produit au niveau du serveur S_8^2, celui-ci n'est plus alimenté (erreur) et ne rend plus le service s_2. Dans le contexte du système global, la défaillance du serveur S_8^2 constitue l'activation d'une faute. On suppose ici que le serveur S_2^2 prend le relais du serveur S_8^2, il délivre le service s_2 à l'utilisateur 5. Deux cas se présentent :

- *Si ce service est délivré sans perte de qualité, il n'y a pas de défaillance.*
- *À l'inverse, une défaillance est présente s'il y a une dégradation de qualité. De plus, les entités $1, 2, 3, 4, 6$ perçoivent la même défaillance. Il s'agit d'une défaillance cohérente.*

Considérons enfin l'ensemble des serveurs S_8^2 et S_2^2 comme un sous-système du système considéré. Dans les mêmes circonstances, si une panne électrique (faute) se produit au niveau du serveur S_8^2, celui-ci n'est plus alimenté (erreur) et ne rend plus le service s_2. Dans le contexte du système global, la défaillance du serveur S_8^2 constitue l'activation d'une faute. On suppose ici que le serveur S_2^2 prend le relais du serveur S_8^2, il délivre le service s_2 à l'utilisateur 5. Deux cas se présentent :

- *Si ce service est délivré sans perte de qualité, il n'y a pas de défaillance.*
- *À l'inverse, une défaillance est présente s'il y a une dégradation de qualité. De plus, les entités $1, 2, 3, 4, 6$ perçoivent la même défaillance. Cependant, l'entité 10 dont le service est délivré par le serveur S_2^2 ne perçoit pas de variation de qualité. Il ne s'agit donc pas d'une défaillance cohérente.*

1.2 Modèles de défaillances considérés

Nous nous intéressons dans le cadre de ce document à la supervision d'un sous-ensemble des entités du système. On sépare ces entités en deux sous-ensembles : d'un

coté les entités supervisées, qui consomment des services fournis par le système tels que VoIP, VOD, ..., et de l'autre les entités qui sont impliquées dans le fonctionnement ou l'acheminement du service délivré. Ce second ensemble contient les liens de communication, les serveurs, les routeurs réseau, etc... et est appelé *environnement*. Dans la figure 1.2, les entités supervisées sont représentées par les ronds bleus et l'environnement par le nuage central.

Nous considérerons dans ce document que les fautes activées au sein de l'environnement peuvent conduire à des défaillances temporelles et cohérentes. De plus, nous supposerons que ces défaillances vérifient un ensemble de contraintes décrites au chapitre 3. La validité de cette hypothèse sera discutée lors de l'évaluation de l'approche décrite dans ce document au chapitre 4.

D'autre part, nous considérons dans les chapitres 3, 4 et 5 que les entités supervisées sont sujettes à des fautes donnant lieu à des défaillances de même nature : des défaillances temporelles et cohérentes.

Nous supposons dans ce document que les entités supervisées effectuent régulièrement des mesures de qualité des services consommés. Lorsqu'une faute est activée, sur une entité supervisée ou au sein de l'environnement, ces mesures varient de façon anormale. Nous supposons l'existence d'un mécanisme permettant de détecter ces variations anormales. Nous pouvons envisager deux types d'approches pour la mise en œuvre pratique de ce mécanisme de détection : les outils de détection de changement ou les outils prédictifs. Ces méthodes supposent qu'en l'absence de défaillances, les valeurs mesurées sont représentatives d'une variable aléatoire de loi inconnue. La présence d'une défaillance se traduit alors par un changement de distribution des valeurs mesurées.

La détection de changement est un outil d'analyse séquentielle visant à détecter les changements dans la distribution de valeurs d'une série temporelle. Lorsque ces mesures varient subitement, ces valeurs ne suivent plus cette distribution. Les outils de détection de changement permettent de détecter cette variation. Un tel mécanisme peut être par exemple mis en œuvre au moyen de Cusum (Cumulative Sum control chart) [Pag54].

Les modèles prédictifs, quant à eux, estiment à chaque instant la valeur de la future mesure. Une différence trop importante entre la mesure effectuée à l'instant suivant et la mesure prédite constitue une défaillance. Les méthodes de lissage exponentiel [Hol04, Win60] qui calculent une moyenne (potentiellement pondérée) sur les dernières valeurs mesurées et l'utilisent comme prévision ou les filtres de Kalman [Kal60] qui permettent d'estimer l'état d'un système dynamique à partir d'une série de mesures incomplètes ou bruitées sont des méthodes prédictives.

Les chapitres 6 et 7 sont dédiés à la modélisation probabiliste et l'évaluation de PeerCube qui constitue une partie du système proposé. Afin d'évaluer la pertinence de cette approche, nous considérons un modèle de défaillances plus fort que celui considéré au début de ce document. Dans ces chapitres, nous considérons un adversaire contrôlant une partie des entités supervisées du système. Ces entités agissent en collusion et sont sujettes à des défaillances byzantines. De cette manière, nous évaluons ce système dans un pire cas.

1.3 Système large échelle

Cette section définit la notion de système large échelle et présente les principes généraux qui régissent la conception de ces systèmes ainsi que les problématiques associées.

Un système est dit "large échelle" s'il est composé d'un très grand nombre d'entités. On dit qu'un système "passe à l'échelle", s'il reste capable de délivrer le service pour lequel il a été conçu, même lorsque le nombre d'entités qui le compose devient très important. Plus précisément, un système large échelle est un système dans lequel les entités interagissent au moyen d'algorithmes locaux. Ce type d'algorithmes ne dépend que de l'état de l'entité ainsi que de son voisinage proche. Cette notion de voisinage varie suivant les systèmes. Un tel système passe à l'échelle si les complexités spatiales (stockage, mémoire, ...) et temporelles (nombre de messages, temps de réponses, ...) des algorithmes exécutés par les entités sont sous-linéaires en fonction du nombre d'entités présentes dans ce système.

Les systèmes pair-à-pairs sont un exemple de système large échelle dans la mesure où il n'est pas inhabituel qu'ils soient composés de plusieurs millions d'entités. Le partage de fichiers est une application bien connue basée sur ce type de système. Dans ce type d'application, chaque entité joue à la fois de rôle de client et de serveur. L'augmentation du nombre d'entités impliquées dans le système contribue à faciliter les échanges de fichiers. Ce type de système passe à l'échelle.

Un protocole est un ensemble de règles régissant les traitements effectués par les entités ainsi que la communication entre ces entités en vue de rendre le service pour lequel le système est conçu. Afin que le système passe à l'échelle, il est nécessaire que le protocole considéré utilise des algorithmes locaux. Les entités du système sont organisées logiquement afin que chacune est accès à son voisinage et non à l'intégralité du système. Une architecture ou overlay désigne la structure logique issue de cette organisation logique, permettant ainsi de définir le voisinage d'une entité. On distingue trois enjeux dans la conception de systèmes large échelle :

1. la gestion du grand nombre d'entités dans le système
2. la gestion du dynamisme des entités dans le système
3. la gestion des fautes dans le système

Gestion de la population Afin d'assurer la propriété de passage à l'échelle, les protocoles de ce type de système ne s'appuient que sur un sous-ensemble des entités du système appelé voisinage. Un protocole définit une structure de réseau logique. Ce réseau constitue le substrat nécessaire à la mise en œuvre du service pour lequel le système est mis en place. On distingue classiquement deux familles de protocoles : les protocoles à architecture non structurée et ceux à architecture structurée.

Dans le cas des architectures non structurées, le lien logique qui unit deux entités du système n'a pas de sémantique. Ces protocoles engendrent généralement un graphe de communication proche d'un graphe aléatoire. Les overlays non structurés sont généralement employés dans le cas de protocoles d'agrégation de données ou afin

de gérer simplement une grande population d'entités. En revanche, ces protocoles ne sont pas adaptés à la recherche de données. En effet, en l'absence de sémantique sur les liens logiques unissant les entités du système, il n'est pas possible d'effectuer une recherche déterministe dans ce type de structure. Par conséquent, la localisation d'une entité se fait par inondation nécessitant ainsi un grand nombre de communications.

À l'inverse, dans le cas des architectures structurées, une sémantique définit le lien unissant deux entités du système. Ces protocoles engendrent généralement des graphe de communication proches d'un graphe régulier. L'exemple le plus courant d'architecture structurée est la table de hachage distribuée (DHT). Dans ce type d'architecture, chaque entité se voit attribuer un identifiant unique de m bits issu d'une fonction de hachage (par exemple MD5 [Riv92] ou SHA-1 [EJ01]). Une entité est alors connectée à une autre si ces deux entités satisfont une condition particulière définie par le protocole concerné. L'espace des identifiants $\{0,1\}^m$ est partitionné entre toutes les entités du système. De la même manière que pour les entités, un identifiant unique de m bits est attribué à chaque donnée gérée par le système. Chaque donnée est gérée par l'entité la plus proche de son identifiant, au sens d'une distance définie par le protocole. Les DHT offrent ainsi une association (clé, valeur) mise en œuvre par les entités présentes dans le système en utilisant les propriétés du graphe de communication. Dans ce document, nous nous concentrerons sur ce type d'architecture.

Gestion du dynamisme Les entités d'un système large échelle peuvent avoir tendance à entrer et sortir du système. Néanmoins ces entrées / sorties ne constituent pas des défaillance par arrêt, ce type de comportement est usuel dans un système large échelle. À chaque entrée ou sortie d'une entité du système, le réseau logique d'interconnexion des entités doit être mis à jour afin d'éviter un partitionnement du réseau logique ou des incohérences. Cette opération de mise à jour peut s'avérer coûteuse et il est donc nécessaire que les protocoles employés gèrent ces mises à jour de manière de manière locale afin de conserver la propriété de passage à l'échelle.

Gestion des comportements fautifs Les entités qui composent un système large échelle sont susceptibles de subir des défaillances byzantines. Dans ce cas, l'entité n'exécute plus le protocole défini et exhibe un comportement arbitraire. Par conséquent, une telle entité peut fournir des informations erronées aux entités de son voisinage pouvant mettre en péril la logique du réseau d'interconnexion des entités ou l'exécution du protocole défini par les entités de son propre voisinage. La gestion des comportements fautifs constitue un élément clé de la conception de ces systèmes.

Bibliographie

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell et C. Landwehr : Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, janvier 2004.

[EJ01] D. Eastlake et P. Jones : US Secure Hash Algorithm 1 (SHA1). Rapport technique, IETF, 2001.

[Hol04] C. C. Holt : Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[Kal60] R. E. Kalman : A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45, 1960.

[Lap96] J. C. Laprie : *Guide de la sûreté de fonctionnement*. Cépaduès-Editions, 1996.

[Pag54] E. S. Page : Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, juin 1954.

[Riv92] R. Rivest : The MD5 Message-Digest Algorithm. Rapport technique, IETF, 1992.

[Win60] P. R. Winters : Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 6:324–342, 1960.

Chapitre 2

Travaux connexes

Nous nous intéressons dans le cadre de cette thèse à distinguer les fautes massives, donnant lieu à une défaillance perçue par un grand nombre d'entités du système, des fautes isolées amenant à une défaillance perçue par un nombre restreint d'entités. Cette distinction est fait dans le cadre d'un réseau large échelle et se base sur la perception de la qualité des services consommés par les entités supervisées sans connaissance de l'environnement.

Nous présentons dans ce chapitre les approches employées dans la conception des systèmes large échelle. Nous présentons ensuite diverses approches visant à regrouper les données similaires. Enfin, nous présentons les travaux existants majeurs sur la détection de fautes massives ou isolées dans les systèmes large échelle.

2.1 Systèmes large échelle

Comme nous l'avons vu au chapitre précédent, on distingue trois enjeux dans la conception de systèmes large échelle :

1. la gestion du grand nombre d'entités dans le système,
2. la gestion du dynamisme des entités dans le système,
3. la gestion des fautes dans le système.

Cette section détaille les principales approches employées pour la conception de systèmes large échelle.

2.1.1 Gestion de la population

Les tables de hachage distribuées (DHT) sont largement utilisées pour la conception de systèmes large échelle. Ces DHT permettent pour organiser logiquement les entités du système afin que chacune d'elles ait accès à un sous-ensemble des entités du système (ce sous-ensemble est appelé voisinage) et non au système dans son intégralité. De cette manière, la propriété de passage à l'échelle est assurée. Il existe un grand nombre de DHT, cette partie présente succinctement deux d'entre elles : CAN [RFH+01], conçu en 2001 par Ratnasamy, Francis, Handley, Karp et Shenker, et Chord [SMK+01], conçu

en 2001 par Stoica, Morris, Karger, Kaashoek et Balakrishnan. Ces DHT font partie des trois DHT originelles proposées en 2001 : Chord, CAN et Pastry [RD01].

Content Addressable Network [RFH+01] (CAN) Dans CAN, les entités sont placées dans un espace virtuel à d dimensions ayant une structure torique. Chaque entité est responsable d'une zone de cet espace. Un lien logique relie les entités gérant des zones voisines dans cet espace, assurant ainsi la connectivité de l'ensemble du système.

Lorsqu'une nouvelle entité rejoint le système, celle-ci contacte l'entité responsable de la zone qu'elle doit occuper. Cette zone est séparée en deux nouvelles zones que chacune des entités va gérer. À l'inverse, lorsqu'une entité quitte le système, la zone dont elle avait la charge est transférée à l'entité la plus proche. Si les deux zones sont voisines et de même surface, celles-ci sont fusionnées. La recherche de données dans ce système se fait de proche en proche en contactant l'entité voisine la plus proche de la destination comme l'illustre la figure 2.1. Dans le cas d'un système composé de n entités, une recherche nécessite de contacter en moyenne $\mathcal{O}\left(n^{1/d}\right)$ entités afin de trouver la donnée ou la zone recherchée.

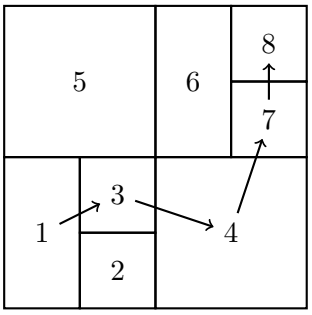


FIGURE 2.1 – Illustration de CAN pour $d = 2$

Chord [SMK+01] De manière similaire, Chord organise ses entités au moyen de leur identifiant à m bits. Ces identifiants sont issus de la fonction de hachage SHA-1 ($m = 160$). Les entités sont ordonnées sur un anneau unidimensionnel. Chaque entité gère alors la portion de l'anneau entre sa position sur celui-ci et l'entité suivante dans l'anneau. À la différence de CAN qui relie les entités voisines dans l'espace virtuel, Chord connecte des entités à des distances variables. Ainsi, si une entité a pour identité p, alors elle sera reliée à chaque entité gérant les positions $p + 2^i, 0 < i < 160$ dans l'anneau. Ces entités forment le voisinage de l'entité p.

La figure 2.2 illustre un anneau Chord pour $m = 4$. On a $n = 16$ entités dans le système. L'entité 0 est liée aux entités $1, 2, 4$ et 8. Ces liens sont représentés par les flèches pleines. Lors d'une recherche de l'entité 11, l'entité 0 recherche dans son voisinage, l'entité qui précède 11 dans le sens horaire de l'anneau : l'entité 8. Celle-ci réitère la recherche, contacte l'entité 10 qui contacte à son tour l'entité 11. Le parcours de cette recherche est représenté par les flèches pointillées. Dans un système comportant

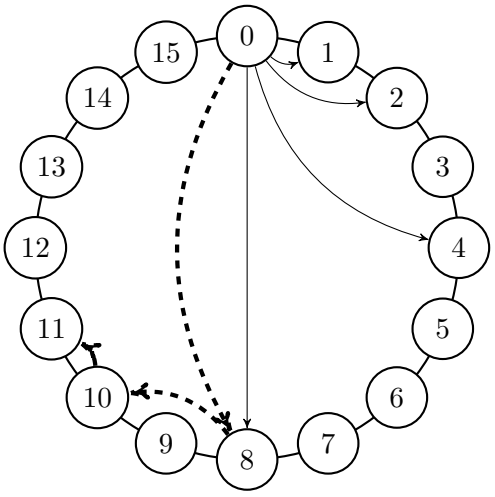


FIGURE 2.2 – Illustration de l'anneau Chord pour $m = 4$

n entités, l'utilisation de liens longs permet de définir une notion de voisinage différente du voisinage physique sur l'anneau. De cette manière, une recherche peut traverser la moitié de l'anneau en un saut et permet ainsi une recherche efficace d'une donnée en contactant $\mathcal{O}(\log n)$ entités en moyenne.

2.1.2 Gérer le dynamisme de la population

Les DHT sont parfaitement adaptées pour gérer de large populations d'entités. De plus elles sont auto-organisantes : l'espace des identifiants est partitionné dynamiquement entre les entités présentes dans le système. D'autre part, ces protocoles offrent des complexités de communication logarithmiques. Cependant, en présence de churn (fréquence des entrées / sorties) important, la topologie devient très dynamique et de nombreuses mises à jours sont nécessaires. Des mise à jours trop fréquentes peuvent amener un problème de cohérence au niveau des entités. Dans le pire des cas, la topologie peut être scindée en plusieurs composantes connexes créant ainsi différentes DHT séparées. Cette partie présente succinctement deux DHT résilientes aux dynamisme des entités dans le système : d'une part Kademlia [MM02], conçu en 2002 par Maymounkov et Mazières, et d'autre part eQuus [LSW06], proposé par Locher, Schmid et Wattenhofer en 2006.

Kademlia [MM02] Dans Kademlia, la sémantique du lien unissant deux entités est plus lâche que dans les DHT telles que CAN ou Chord. Une entité dont l'identifiant à m bits vaut $b_0 b_1 \dots b_{m-1}$ est liée à m ensembles de k entités appelés k-buckets. Le premier bucket contient k entités dont l'identifiant débute par $\overline{b_0}$, le second par $b_0\overline{b_1}$, le i-eme par $b_0 b_1 \dots \overline{b_{i-1}}$.

Une entité de Kademlia met à jour son i-ème k-bucket dès qu'elle transfère un

message provenant d'une entité dont l'identifiant débute par $b_0 b_1 \dots \overline{b_{i-1}}$. Ainsi, chaque k-bucket contient les k dernières entités du système vues par l'entité courante. De cette manière, chaque entité profite des messages qu'elle fait suivre pour garder une vue à jour des entités effectivement présentes dans le système. L'entité réduit ainsi le nombre de messages de mise-à-jour pour maintenir la cohérence de la topologie.

De la même manière que Chord, Kademlia emploie des liens de plus en plus éloignés de l'entité courante. Une recherche dans cette architecture nécessite de contacter $\mathcal{O}(\log n)$ entités, dans un système comportant n entités.

eQuus [LSW06] eQuus emploie une méthode différente pour gérer le dynamisme des entités. Contrairement aux architectures précédentes dans lesquelles chaque entité gère une partie de l'espace des identifiants, eQuus attribue cette gestion à un ensemble d'entités appelé clique.

Une clique est un ensemble d'entités dont la taille est comprise entre deux paramètres du système $\mathcal{L}$ et $\mathcal{U}$. De plus, chaque clique est identifiée de manière unique par un identifiant i de $d \ll m$ bits, tel que toutes les entités de la clique aient leur identifiant préfixé par i. Lorsqu'une clique devient sur-peuplée, deux nouvelles cliques sont créées en repartissant les entités entre elles. À l'inverse, lorsqu'une clique devient sous-peuplée, elle est fusionnée avec une clique existante.

La recherche d'une donnée dans eQuus consiste alors à trouver la clique contenant l'entité ou la donnée recherchée. L'utilisation des cliques permet de réduire le diamètre moyen du graphe de communication engendré par le protocole, ainsi que le nombre de liens liant les cliques entre elles. Dans un système comportant n entités, l'utilisation des cliques identifiées par d bits permet ainsi une recherche efficace d'une donnée en contactant $\mathcal{O}(\log_{2^d} n)$ entités en moyenne. Le protocole eQuus absorbe le dynamisme des entités en distinguant deux types de mises-à-jour des informations maintenues par celles-ci. La modification de la composition d'une clique est traitée immédiatement, toutes les entités de la clique mettent à jour la composition de celle-ci. À l'inverse, les entités des cliques voisines reçoivent plus rarement les mises-à-jour. Une recherche dans eQuus consiste à contacter toutes les entités présentes dans une clique. Les fréquences de mise-à-jour intra et inter-cliques peuvent alors amener une différence de compositions. Cette fréquence des mise-à-jour inter-clique est paramétrée de manière à ce que ces compositions soient d'intersection non vide tout en évitant les mises-à-jours systématiques.

2.1.3 Gérer les comportement fautifs

Les protocoles que nous venons de présenter sont capables de gérer de grandes populations d'entités et tolèrent de manière variable le dynamisme des entités au sein du système. Néanmoins, ces protocoles ne fonctionnent qu'en l'absence de fautes byzantines. Une entité sujette à une faute byzantine ou entité byzantine ne suit pas nécessairement le protocole. Elle se comporte de manière arbitraire. Ce comportement peut mettre en péril la connectivité de l'architecture ou amener à des recherches inabouties. Cette partie présente succinctement deux DHT résilientes aux comporte-

ments byzantins : S-chord [FSY05], proposé par Fiat, Saia et Young en 2005, et PeerCube [ABLR08], conçu par Anceaume, Brasileiro, Ludinard et Ravoaja en 2008.

S-Chord [FSY05] Une amélioration de Chord visant à rendre celui-ci tolérant aux fautes byzantines est décrite dans [FSY05]. Pour chaque entité p de l'anneau est défini l'ensemble $S(p)$ contenant les entités q du système dont la distance de q à p n'excède pas $C(\ln(n))/n$, où n désigne le nombre d'entités dans le système et C est un paramètre du système. L'idée de S-Chord est de remplacer toutes les communications entre deux entités p et q par des communications entre les ensembles $S(p)$ et $S(q)$. La figure 2.3 illustre un anneau S-Chord pour $m = 4$ et $C(\ln n)/n = 2$. On a $n = 16$ entités dans le système. Comme dans Chord, l'entité 0 est liée aux entités $1, 2, 4$ et 8. Lors d'une recherche de l'entité 11, l'entité 0 contacte l'ensemble $S(0)$ afin de contacter l'ensemble $S(q)$ de l'entité liée la plus proche de 11 tout en la précédent. L'ensemble $S(0)$ contacte alors l'ensemble $S(8)$. Cet ensemble réitère la recherche contacte l'ensemble $S(10)$ qui contacte à son tour l'entité $S(11)$. Le parcours de cette recherche est représenté par les flèches pointillées. La figure 2.4 illustre toutes les communications engendrées par cette recherche. Cette approche pour tolérer les comportements byzantins est sensiblement plus coûteuse en communications que le protocole Chord classique. En présence de fautes byzantines indépendantes, c'est-à-dire qui ne résultent pas d'entités malveillantes agissant en collusion, S-Chord assure ainsi qu'une recherche dans l'overlay composé de n entités nécessite en moyenne $\mathcal{O}\big(\log^2 n\big)$ messages. Cependant, ce type d'approche engendre $\mathcal{O}\big(\log^3 n\big)$ messages lors des entrées et sorties des entités du système.Cela signifie donc que ce type d'approche n'est pas adapté dans un contexte très dynamique.

PeerCube [ABLR08] De manière similaire, PeerCube remplace la communication entre deux entités du système par une communication entre groupes d'entités. La topologie engendrée par ce protocole s'approche d'un hypercube parfait dans lequel les sommets sont constitués d'un groupe d'entités appelés cluster. Chaque cluster est en charge d'une partie de l'espace des identifiants. Contrairement à S-Chord dans lequel pour chaque entité p du système il existe un ensemble $S(p)$ lui correspondant, PeerCube partitionne l'espace des identifiants entre les clusters. Ainsi plusieurs entités correspondent au même cluster. La communication au sein de l'architecture est assurée par les clusters. Afin de tolérer le churn naturel présent dans les systèmes peer-to-peer, les clusters sont décomposés en deux sous-ensembles *core* et *spare*. Le premier forme une clique de taille constante γ et assure la connectivité du graphe tandis que le second agit comme un tampon et absorbe les entités entrantes dans le cluster de manière transparente vis-à-vis du reste du système. D'autre part, la faible taille γ permet de mettre en œuvre des algorithmes [CFNV04, KAD+07] de consensus tolérants au byzantins afin de gérer les mises à jours de la composition du cluster. Les opérations basiques de cette architecture nécessitent en moyenne $\mathcal{O}(\log n)$ messages, avec n le nombre d'entités présentes au sein du système. Enfin, bien que cette approche soit résiliente à la présence d'entités byzantines, celles ci peuvent prendre le contrôle d'un cluster et mettre en péril l'exécution de recherches dans le système. PeerCube emploie alors une technique de recherche redondante sur des chemins indépendants assurant ainsi un haut taux de succès

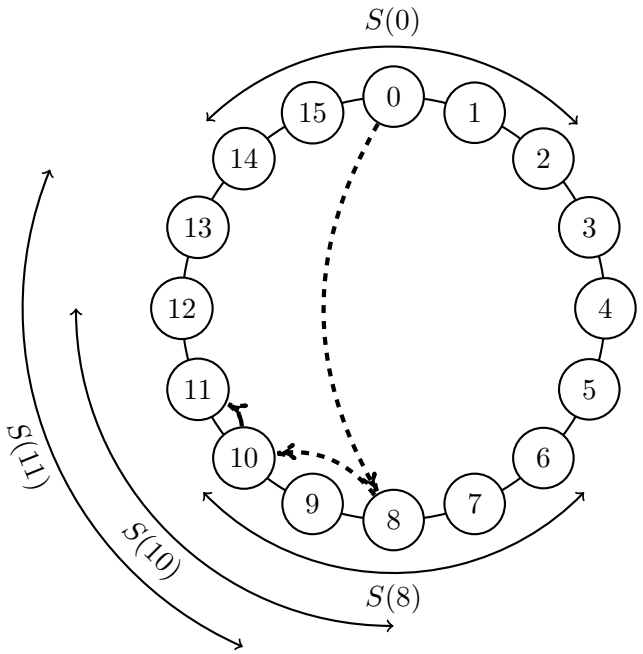


FIGURE 2.3 – Illustration de l'anneau S-Chord pour $m = 4$ et $C(\ln n)/n = 2$

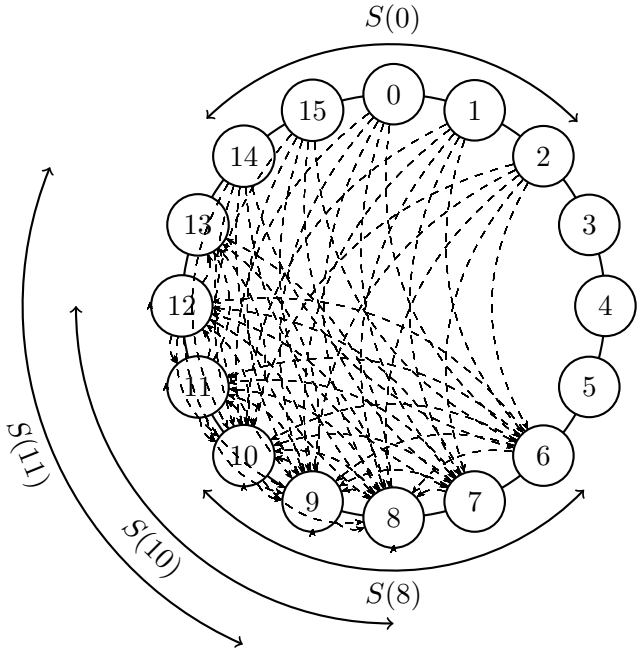


FIGURE 2.4 – Illustration de l'anneau S-Chord pour $m = 4$ et $C(\ln n)/n = 2$

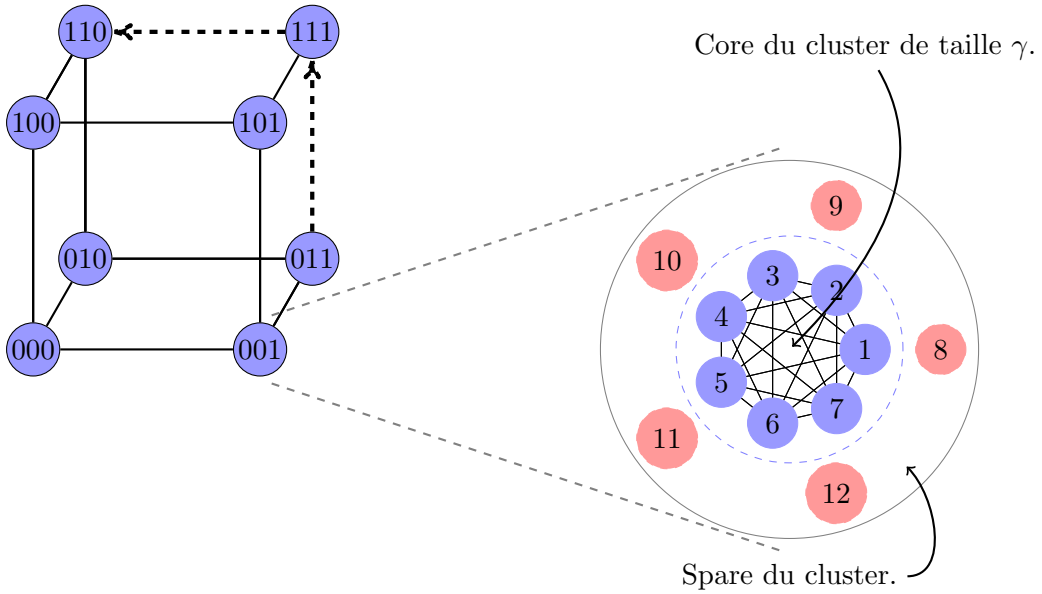


FIGURE 2.5 – Illustration de l'architecture PeerCube. À gauche, l'organisation en hypercube des clusters, à droite la composition d'un cluster particulier.

des recherches malgré la présence de ces entités malveillantes. Une étude approfondie de cette architecture est fournie dans les chapitre 6 et 7 du présent document.

Nous avons présenté dans cette section un aperçu des travaux existants permettant de gérer de grandes population d'entités au sein d'un système, ainsi que leur dynamisme et la présence de fautes impactant ces entités. La section suivante traite des données générées par ces entités et porte plus particulièrement sur les techniques visant à regrouper les données similaires.

2.2 Organisation des données

Les entités d'un système génèrent de manière continue diverses données. Il peut s'agir de données applicatives ou de données relatives à l'état interne des entités. En particulier, nous supposons dans le cadre de ce document que les entités effectuent régulièrement des mesures de qualités des services consommés. De plus, nous supposons que l'activation d'une faute dans le système se traduit par une variation de ces mesures de qualité.

Nous nous intéressons dans le cadre de ce document à distinguer les fautes présentes dans le système suivant le nombre d'entités percevant les défaillances qui en résultent. Plus précisément, nous nous intéressons aux fautes ayant un impact sur la qualité des services consommés par les entités supervisées. Nous faisons ici l'hypothèse suivante : l'activation d'une faute dans le système a un impact similaire sur la qualité du service perçue par les entités supervisées. Afin de distinguer une faute menant à une défaillance

perçue par un grand nombre d'utilisateurs de celle perçue par un petit nombre d'entre eux, il est nécessaire de regrouper ensemble les variations similaires de qualités. On peut détecter les perceptions similaires en partitionnant les entités supervisées en fonction des qualités mesurées par celles-ci.

Nous présentons dans cette section divers travaux existants permettant de regrouper des données similaires. On distingue deux approches principales. Les approches basées sur le partionnement des données statiques sont présentées dans la sous-section 2.2.1. Les approches travaillant sur des données dynamiques sont présentées dans la sous-section 2.2.2.

2.2.1 Partitionnement de données statiques

La clustering est la tâche qui consiste à diviser un ensemble de données D en sous-ensembles appelés clusters de sorte que chaque donnée d'un sous-ensemble soit regroupée avec les données qui lui ressemblent le plus.

La similarité est le critère central pour l'identification de clusters dans l'ensemble de données D. On distingue différentes notions de similarité qui engendrent plusieurs familles de méthodes de partitionnement. S'il existe une distance δ entre les données de D, celle-ci peut être utilisée pour partitionner l'ensemble D.

Partitionnement hiérarchique Les méthodes à partitionnement hiérarchique cherchent à répartir les données d'un ensemble D parmi κ clusters, κ étant un paramètre d'entrée de l'algorithme. Initialement, ce type d'algorithme attribue à chaque donnée x l'ensemble $\{x\}$. Ensuite, à chaque étape, l'algorithme fusionne les ensembles les plus proches au sens d'une distance Δ sur ces ensembles, et s'arrête lorsqu'il ne reste que κ ensembles. La différence principale entre les différents algorithmes à partitionnement hiérarchique se situe dans la mesure de distance entre ensemble de données. Par exemple, l'algorithme SLINK [Sib73] fusionne deux ensembles X, Y lorsque la distance Δ inter-ensemble $\Delta(X,Y) = \min_{x\in X, y\in Y} \delta(x,y)$ est minimale. À l'inverse, l'algorithme CLINK [Def77] cherche à minimiser le diamètre de la réunion des ensembles X, Y en minimisant $\Delta(X,Y) = \max_{x\in X, y\in Y} \delta(x,y)$. La complexité de ces algorithmes est $\mathcal{O}\big(|D|^2\big)$, ces algorithmes sont donc trop coûteux pour être utilisés dans le contexte des systèmes large échelle.

Partitionnement orienté distance Les méthodes à partitionnement hiérarchique nécessitent de travailler sur l'ensemble des données D induisant ainsi une complexité de calcul importante. Afin de réduire cette complexité, des algorithmes [HW79, DLR77] travaillant sur un sous-ensemble des données ont été proposés.

De la même manière que les méthodes à partitionnement hiérarchique, le but de l'algorithme des κ-moyennes [HW79] est de partitionner les données de l'ensemble D en κ sous-ensembles. Cet algorithme est identique à l'algorithme de quantification de Lloyd [DLR77]. Les données de D sont placées dans un espace métrique E. L'algorithme est initialisé avec un ensemble $\Sigma = \{\sigma_1, \ldots, \sigma_\kappa\}$.

Deux méthodes principales, décrites dans [HE02], existent pour l'initialisation de l'algorithme des κ-moyennes. La méthode aléatoire consiste à choisir uniformément κ données de D afin de construire l'ensemble initial Σ. À l'inverse, la méthode Forgy consiste à choisir uniformément κ points de l'espace E.

L'algorithme associe ensuite à chaque donnnée $x \in D$, l'élément de Σ le plus proche au sens de la distance δ. Les ensembles Σ_i sont définis par :

$$\forall i \in \{1, \ldots, \kappa\}, \Sigma_i = \{x \in D \mid \delta(x, \sigma_i) \leq \delta(x, \sigma_j) \forall j \in \{1, \ldots, \kappa\}\}.$$

Ces ensembles forment une partition de D. Les éléments de Σ sont replacés à la position de l'isobarycentre des entités de Σ_i. On a :

$$\forall 1 \leq i \leq \kappa, \sigma_i = \frac{1}{|\Sigma_i|} \sum_{j \in \Sigma_i} j.$$

Ce processus est répété jusqu'à ce que les éléments de Σ soient stables.

Trouver un partitionnement d'un ensemble de données D en κ clusters se fait en appliquant l'algorithme des κ-moyennes. En revanche, trouver la partition qui minimise la distance au sein des parties est un problème NP-difficile. Un problème NP-difficile est un problème de décision pour lequel on ne sait pas vérifier une solution en temps polynomial.

Par conséquent, on utilise des heuristiques pour le critère d'arrêt de l'algorithme. Une solution courante consiste à fournir un nombre I d'itérations et de considérer les parties Σ_i comme une partition valable. Dans un espace de dimension d, la complexité de cet algorithme est $\mathcal{O}(Id\kappa|D|))$, avec I le nombre d'itérations, κ le nombre sous-ensembles de la partition et D l'ensemble des données initial. Bien que cet algorithme ait une complexité linéaire en nombre de données traitées, celui-ci est facilement parallélisable.

D'autre part, l'algorithme heuristique construit itérativement une partition minimisant la distance au sein des clusters. Cet algorithme converge donc vers un minimum local. Afin de pallier le biais introduit pas le choix initial de l'ensemble Σ, cet algorithme est exécuté à diverses reprises afin d'éviter un minimum local et ne conserver que la meilleure partition finale. L'algorithme des κ-moyennes est détaillé dans le chapitre 5.

Différentes variantes de cet algorithme ont été proposées. À la différence de l'algorithme des κ-moyennes qui calcule les ensembles $\Sigma_i, 1 \leq i \leq \kappa$ centrés sur l'isobarycentre de cet ensemble, l'algorithme κ-medoids [KR87] centre ces ensembles sur une donnée $x \in D$. De plus, cet algorithme utilise la norme 1 comme distance, le rendant plus robuste à la présence de données anormales.

L'algorithme des c-moyennes [BEF84] quant à lui n'effectue pas un partitionnement de l'ensemble des données D. À l'inverse, cet algorithme attribue à chaque donnée $x \in D$, un degré d'appartenance à chaque ensemble $\Sigma_i, 1 \leq \kappa$.

L'inconvénient majeur de ces approches est le paramétrage de κ. En effet, il n'est pas possible de connaître a priori la valeur idéale de ce paramètre. Des méthodes [PM00] visent à estimer la valeur idéale de ce paramètre pour l'ensemble de données D considéré, nécessitant ainsi un prétraitement de l'ensemble des données.

Partitionnement orienté densité Afin de s'affranchir des contraintes d'initialisation et du nombre κ de clusters recherchés, d'autres approches ont été envisagées. L'algorithme DBSCAN [EKSX96] divise l'ensemble de données D en sous-ensembles de forte densité séparés par des espaces de faible densité. L'algorithme DBSCAN est détaillé dans le chapitre 4. Cet algorithme requiert deux paramètres, ε et m, et une distance δ. Le paramètre ε permet de définir le voisinage d'une entité tandis que le paramètre m définit un seuil de densité sur ce voisinage. Deux données $x, y \in D$ sont voisines si $\delta(x, y) \leq \varepsilon$. Le voisinage $N_\varepsilon(x)$ d'une donnée $x \in D$ est dense s'il contient plus de m données, la donnée x est alors dite centrale. L'algorithme DBSCAN parcourt l'ensemble des données D. Pour chaque donnée centrale non visitée, l'algorithme crée un nouveau sous-ensemble et lui ajoute cette donnée ainsi que son voisinage. Si ce voisinage contient des données centrales, le voisinage de celles-ci est ajouté au sous-ensemble courant. Les données sont ainsi traitées de proche en proche et le calcul du sous-ensemble courant s'arrête lorsque toutes les données centrales et leur voisinage ainsi atteignables ont été traitées. Cet algorithme nécessite de traiter toutes les données et pour chacune d'elle de construire son voisinage. Par conséquent, la complexité de cet algorithme est $\mathcal{O}\big(|D|^2\big)$. L'utilisation de structure d'indexation de données spatiales du type R*-tree [BKSS90] permet de réduire cette complexité à $\mathcal{O}(|D| \log |D|)$.

À la différence de l'algorithme des κ-moyennes qui nécessite de spécifier le nombre de sous-ensembles recherchés, DBSCAN construit lui-même le nombre de sous-ensembles nécessaires. De plus, il n'y a pas de problème d'initialisation, toute exécution de l'algorithme mène globalement aux mêmes sous-ensembles : une seule exécution de l'algorithme est suffisante pour les construire. Cependant, il arrive que deux exécutions attribuent une même donnée à des sous-ensembles différents. Cela arrive lorsqu'une donnée n'est pas centrale mais qu'elle appartient à l'intersection des voisinages de deux données centrales appartenant à des sous-ensembles différents. Suivant l'ordre de traitement des données, celle-ci est classée dans l'un ou l'autre des sous-ensembles.

L'algorithme DBSCAN définit une notion de densité fonction de deux paramètres ε et m et construit des clusters à partir de celle-ci. Si l'ensemble des données D est constitué de divers sous-ensembles de densités différentes, DBSCAN échoue à les retrouver. D'autres approches, par exemple OPTICS [ABKS99], ont été proposées afin de pallier cette limite. OPTICS suit la même idée que DBSCAN en distinguant les données centrales. Cependant, en introduisant une seconde distance ε' définie comme la plus petite distance entre une donnée $x \in D$ et une donnée $y \in N_\varepsilon(x)$, OPTICS peut gérer des sous-ensembles de densités différentes ε'/m. D'autre part, GDBSCAN [SEKX98] est une généralisation de DBSCAN. Dans GDBSCAN, les deux paramètres ε et m sont remplacés par des fonctions plus générales définissant la notion de voisinage ainsi que la notion de densité. Cette généralisation permet alors de partitionner des données pour lesquelles il n'y a pas de notion de distance.

2.2.2 Données dynamiques

Initialement, les techniques de clustering ont été utilisées sur des données statiques. Elles ont été ensuite étendues sur des données ayant une évolution temporelle. Plutôt

que de considérer les données à chaque instant et d'y appliquer les techniques de clustering que nous venons de voir, les approches [JYZ+08, JLO07] étudient l'évolution, l'apparition et la disparition des clusters au cours du temps en fonction de l'évolution temporelle des données.

Cependant, comme nous l'avons vu, le clustering cherche à regrouper des données similaires. La notion de similarité est centrale dans ce regroupement. Aucun des travaux que nous avons présenté ne borne l'écart de similarité entre deux données d'un même sous-ensemble. Par exemple, dans le cas de l'algorithme κ-moyennes, une donnée $x \in D$ est associée au plus proche élément $\sigma_i, 1 \leq i \leq \kappa$. C'est donc la position des σ_i qui détermine la distance maximale entre deux éléments d'un même cluster Σ_i. Ainsi, la distance entre deux données $x, y \in \Sigma_i$ peut être arbitrairement grande.

De la même manière dans le cas des algorithmes basés sur la densité des données, deux données appartiennent à un même cluster s'il existe suffisamment de données centrales entre elles de sorte que ces données soient connectées de proche en proche par un chemin dense. Cette fois encore, deux données peuvent donc appartenir à un même cluster tout en ayant une similarité très faible.

L'utilisation de la notion de groupe est une autre approche visant à regrouper des données dynamiques. Un groupe est défini comme un ensemble contenant plus de τ données dont la distance maximale entre deux données n'excède pas un paramètre fixé r dans un intervalle de temps donné. Ces travaux portant sur la recherche de groupe tels que [BGHW06, BGHW08, BBG08, VBT09], s'intéressent à exhiber les données formant un groupe le plus longtemps possible sans contrainte de partitionnement des données. La recherche de mouvements de groupes décrite se différencie du clustering par trois aspects.

1. Ce type d'approche s'intéresse à exhiber l'ensemble des groupes auquel une donnée appartient, sans cependant imposer le partitionnement (appartenance à un unique cluster) présent dans le clustering.
2. À l'inverse du clustering, où l'on cherche à construire des clusters, la recherche de mouvements de groupes est centrée sur les données et leur potentielle appartenance à différents ensembles.
3. Des données ayant une similarité trop faible ne peuvent être regroupées ensemble. On observe alors à des instants discrets l'ensemble des données D et celles-ci sont regroupées en sous-ensembles dont la distance maximale entre deux éléments n'excède pas un seuil prédéfini r à chaque instant discret.

Nous avons présenté dans cette section les différents travaux existants permettant de regrouper ensemble des données similaires d'un ensemble de données. La section suivante présente l'utilisation de ces techniques dans le contexte de la supervision d'entités et la caractérisation de fautes.

2.3 Supervision

Nous avons présenté dans les sections précédentes différents travaux existants permettant de gérer un grand ensemble d'entités ainsi que les données qu'elles sont sus-

ceptibles de générer. Nous présentons à présent l'utilisation de ces techniques dans le contexte de la supervision d'entités et la caractérisation de fautes.

2.3.1 État global du système

Les entités supervisées génèrent de manière continue un ensemble de données qui sont ensuite mises en forme au sein d'une interface de supervision au niveau d'une entité appelée superviseur. Ces données sont variables et peuvent par exemple représenter l'état interne de l'entité, une qualité de service, etc...

Les systèmes de supervision sont des éléments clés pour la caractérisation des fautes dans le système. Le but de ces systèmes est de rendre compte de manière continue de l'état du système considéré. Il est donc nécessaire pour ces systèmes d'être capable d'appréhender le système supervisé dans son ensemble, de manière réactive et avec un faible surcoût afin de ne pas perturber le bon fonctionnement du système supervisé.

On distingue alors deux types d'approches dans la supervision : celles fournissant une vue agrégée du système et celles cherchant à avoir une vision de l'état de chacun des éléments du système supervisé. Astrolabe [VRBV03] et SDIMS [YD04] organisent de manière distribuée les entités du système supervisé. Les entités sont organisées de manière hiérarchique en fonction des données générées qu'elles génèrent. Cette hiérarchie permet alors de collecter et d'agréger les informations relatives aux entités du système. De cette manière, ces systèmes fournissent une vue résumée de la santé de l'ensemble du système. Les travaux décrits dans ce document portent sur la supervision d'un ensemble d'entités et la caractérisation de fautes, ces approches ne sont donc pas adaptées dans notre contexte.

À l'inverse, Ganglia [MCC03], CoMon [PP06] et InfoTrack [ZTG+09] supervisent chacune des entités du système. Ces approches exploitent les similarités entre les données générées par les entités supervisées afin de réduire la communication entre les entités et l'interface de supervision. Par exemple, InfoTrack repose sur deux principes : autocorrélation et clustering. Les entités supervisées sont regroupées en κ clusters (issus de l'exécution de l'algorithme κ-moyennes) par rapport à une mesure de similarité sur les données générées.

Au sein de chaque cluster, l'entité dont les données générées sont les plus proches de la valeur médiane des données générées par l'ensemble des entités du cluster prend en charge la communication avec l'extérieur du cluster. Cette entité est appelée leader du cluster. Afin de diminuer la communication entre les entités, celles-ci sont munies d'outils de prédiction [Kal60]. Les leaders sont munis du même outil de prédiction. Ainsi, à chaque instant le leader a accès aux même prédictions que chacune des entités du cluster. Par conséquent, lorsque la donnée effective est proche de la donnée prédite, aucune communication entre l'entité et le leader n'est nécessaire. Dans le cas contraire, l'entité communique la donnée effective au leader. Le même mécanisme est mis en place entre les leaders de chaque cluster et le superviseur.

Le leader maintient de plus un compteur pour chaque entité du cluster. Ce compteur est incrémenté lorsque le leader effectue une prédiction en accord avec la donnée générée par l'entité, et décrémenté dans le cas contraire. Lorsque le compteur d'une

entité est inférieur à un seuil prédéfini θ_1, les données qu'elle génère ne sont plus en adéquation avec les données générées par les autres entités du cluster. Cette entité est alors replacée dans un autre cluster plus approprié. À l'inverse, si un nombre θ_2 d'entités du cluster doivent être replacées, cela signifie que le leader n'est plus représentatif des entités du cluster. Un nouveau leader est alors choisi. Ces opérations de maintenance sont effectuées de manière centralisée par le superviseur. Les résultats expérimentaux montrent ainsi une diminution des coûts de communication de 50 à 90% tout en conservant une haute à précision (de 1 à 5% d'erreur).

Cependant, ce type d'approche n'est applicables qu'aux mesures locales (comme la charge CPU, consommation mémoire, temps de fonctionnement sans redémarrage, etc...). Ce type de mesure est continu et ne dépend pas de l'activité utilisateur. De plus, la stabilité et prédictibilité de ce type de mesure permet de réduire efficacement les communications nécessaires à la supervision.

L'initialisation de ce type de solution repose sur l'exécution d'algorithme de clustering par le superviseur. Cette exécution initial est effectuée hors-ligne. Le partitionnement ainsi calculé est mis-à-jour lorsqu'une entité doit être replacée. D'autre part, les variations dans ces mesures peuvent être symptomatiques d'une faute au niveau d'une entité supervisée. Ce type d'approche permet donc de détecter des défaillances par le replacement d'une entité mais ne permet pas de caractériser le type de faute sous-jacent. Les défaillances qui impactent un grand nombre d'entités sont donc très coûteuses d'un point de vue algorithmique. L'absence de caractérisation ne fournit donc pas d'indice sur les causes qui ont provoqué la défaillance.

2.3.2 Caractérisation de fautes

Lorsqu'une faute est activée dans le système, celle-ci peut induire une défaillance. Cette défaillance se traduit par une dégradation de la qualité du service rendu par le système. Les outils de supervision que l'on vient de présenter sont destinés à fournir une vision globale de la santé du système. En revanche, ces outils ne permettent pas de localiser la faute qui a impacté le système. La localisation de fautes [MA04] consiste à déduire la cause effective d'une défaillance à partir d'un ensemble d'observations. On distingue trois types d'approches pour la localisation de fautes. Le recours à des systèmes experts qui consiste à mettre en place des règles ou des arbres de décisions ; les modèles de propagation de fautes qui s'appuient sur des graphes de dépendances, et enfin la connaissance du graphe de communication entre les entités.

La connaissance supplémentaire requise dans le cadre de la localisation de fautes n'est pas envisageable dans le cadre de la supervision de systèmes large échelle. Dans le cadre de cette thèse, nous souhaitons caractériser les fautes qui induisent une défaillance dans le système suivant le nombre d'entités qui perçoivent cette défaillance. Cette caractérisation permettra ensuite de réduire la connaissance additionnelle nécessaire à la localisation de fautes.

Tiresias [HCDW12], Argus [YFG+12] et CEM [CBG10] sont trois approches liées à la caractérisation de fautes dans les réseaux large échelle. Ces trois approches reposent sur la perception de défaillances par les utilisateurs du système.

Tiresias Cette approche [HCDW12] classifie des données opérationnelles d'après six étapes. Ces données sont issues des appels des clients au service de support d'un FAI ainsi que de l'analyse des données de journalisation des pannes sur les équipements clients. Ces données sont horodatées et associées à un niveau hiérarchique du réseau. Ces données proviennent de divers endroits dans le réseau, arrivent en masse et de manière continue au niveau du serveur Tiresias. Le serveur maintient une fenêtre temporelle divisée en différents intervalles de temps. Une donnée est affectée à l'intervalle de temps au cours duquel elle a été produite. Chaque donnée est ensuite placée dans un arbre en fonction du niveau hiérarchique qui lui est associé et un poids lui est attribué. Un traitement mathématique est effectué sur les données dont le poids associé est situé au dessus d'un seuil prédéfini θ. Ce traitement permet d'exhiber les données anormales traduisant l'activation d'une faute impactant un grand nombre d'utilisateurs.

Cette approche souffre de trois limitations. Tout d'abord, Tiresias s'appuie sur les appels des utilisateurs et déporte donc la tâche de détection de faute au niveau de l'utilisateur. Il n'y a donc pas de garantie sur le délai entre la détection de la faute par l'utilisateur et la notification au centre de support. D'autre part, chaque donnée est traitée manuellement par un technicien qui choisit le niveau hiérarchique à associer à la donnée opérationnelle. Ce traitement humain est donc source d'erreurs et est incompatible avec un traitement des données à la volée. Enfin, cette approche s'appuie sur la connaissance du réseau d'interconnexion et se limite de plus aux réseaux à structure hiérarchique (comme les réseaux IPTV).

Argus De manière similaire à Tiresias, Argus [YFG+12] a une approche centralisée. Cette approche s'appuie sur le déploiement par les fournisseurs d'accès à Internet de leurs propre réseaux de services CDN, VoIP, IPTV afin de mutualiser la tâche de supervision. Ainsi, Argus tire parti des informations relatives à la topologie réseau, au routage, à la géographie, et aux mesures de performance de bout en bout. Comme la totalité de l'environnement est maîtrisé, Argus peut superviser les mesures de performance du point de vue serveur et non du point de vue du client. Ainsi, les utilisateurs sont clusterisés de manière hiérarchique par rapport à leur localisation dans la topologie réseau. Argus génère ensuite pour chaque groupe d'utilisateurs, une série temporelle correspondant aux mesures de performances agrégées sur le groupe d'utilisateur concerné. Ces séries temporelles sont ensuite utilisées pour détecter des fautes qui impactent tout ou partie du groupe d'utilisateur concerné.

L'approche centralisée d'Argus nécessite de traiter l'ensemble des données générées par l'ensemble des utilisateurs consommant les services proposés au sein du réseau du FAI. Par conséquent, il se pose un problème clair vis-à-vis du passage à l'échelle de cette solution. De plus, Argus adopte un point de vue totalement omniscient vis-à-vis de la caractérisation des fautes qui impactent le système : la topologie, le positionnement des différents serveurs et des clients sont connus et utilisés pour caractériser les fautes.

CEM De la même manière, CEM [CBG10, Cho10] cherche à distinguer les fautes impactant un petit nombre d'entités supervisées des fautes dont la défaillance est perçue

par un grand nombre d'entités. Dans le cas d'une dégradation anormale de la qualité d'un service consommé par une entité, celle-ci incrémente un compteur situé une table de hachage distribuée [MM02]. La clé utilisée pour l'écriture dans la table de hachage distribuée correspond à l'identifiant du système autonome auquel l'entité supervisée appartient. Un Système Autonome est un réseau au sein d'Internet dont la politique de routage est cohérente. Les systèmes autonomes sont interconnectés et forment ainsi Internet. Le réseau d'un fournisseur d'accès à Internet est généralement un Système Autonome. Chaque système autonome est identifié de manière unique par une valeur utilisée lors du routage par BGP (Border Gateway Protocol [RLH06]) entre les différents systèmes autonomes.

Plusieurs entités peuvent percevoir une défaillance mais cela ne suffit pas à assurer qu'il s'agit d'une même défaillance induite par une même faute. Deux cas sont envisageables :

Cas 1 diverses fautes concomitantes et indépendantes mènent aux défaillances perçues par les entités,

Cas 2 une unique faute provoque une défaillance perçue par ces mêmes entités.

Les auteurs de CEM proposent une mesure de vraisemblance notée LR. Cette mesure est utilisée comme un seuil permettant de faire la distinction entre ces deux cas. Cette mesure correspond au rapport $LR = P_e/P_u$ entre la probabilité P_e que n entités distinctes perçoivent des défaillances dues à n fautes distinctes dans un intervalle de temps donné (permier cas) et la probabilité P_u que n entités perçoivent une défaillance due à une même faute dans ce même intervalle de temps (second cas). Une valeur de vraisemblance $LR > 1$ tend à indiquer une défaillance liée au réseau tandis que $LR \leq 1$ tend à indiquer des défaillances individuelles.

Cette approche est ensuite comparée aux données opérationnelles de divers fournisseurs d'accès à Internet. La valeur de LR est centrale dans la caractérisation des fautes impactant le système. Celle-ci est positionnée statiquement à partir des données issues des données opérationnelles des divers fournisseurs d'accès à Internet. Cette approche est ensuite appliquée sur des mesures de qualité collectées à partir de clients BitTorrent. Le résultat de la caractérisation est ensuite comparée à la vision des différents fournisseurs d'accès à Internet pendant la période considérée.

Les résultats de cette comparaison sont difficiles à interpréter. Sur l'ensemble des défaillances que les fournisseurs d'accès à Internet attribuent à des fautes massives, CEM en attribue 80% à des fautes massives. Cependant, cette proportion représente seulement 30% des défaillances attribuées à des fautes massives par CEM. Les opérateurs n'ont pas connaissance des 70% restant de défaillances décelées par cette approche. Par conséquent, il est impossible de savoir s'il s'agit de faux positifs de cette approche ou d'une très forte proportion de défaillances ignorées par les fournisseurs d'accès à Internet.

Bien que cette approche soit une première approche vers la caractérisation des fautes impactant le système en fonction du nombre d'entités percevant la défaillance induite, celle-ci présente des limitations. Tout d'abord, celle-ci requière une connaissance du réseau afin de regrouper ensemble les perceptions des entités d'un même système auto-

nome. Ensuite, le seuil de classification LR est fixé de manière empirique par l'analyse statistique de données issues de différents fournisseurs d'accès à Internet. Enfin, bien que cette approche soit un premier permettant de caractériser une faute en fonction de son impact, la validation de cette approche n'est pas tout à fait convaincante dans la mesure où l'on ne sait pas si les 70% de défaillances supplémentaires décelées constituent de réelles défaillances ou des faux positifs.

Nous avons présenté dans ce chapitre les concepts nécessaires aux travaux décrits dans ce document. De plus, nous avons décrit les techniques classiques utilisées dans les systèmes distribués large échelle ainsi que dans le cadre du clustering de données. Enfin, ces techniques nous ont permis de présenter les travaux connexes existants sur la supervision et la caractérisation de fautes dans les réseaux large échelle.

Bibliographie

[ABKS99] M. Ankerst, M. M. Breunig, H. P. Kriegel et J. Sander : OPTICS : Ordering Points To Identify the Clustering Structure. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

[BBG08] K. Buchin, M. Buchin et J. Gudmundsson : Detecting Single File Movement. Dans *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2008.

[BEF84] J. C Bezdek, R. Ehrlich et W. Full : FCM : The Fuzzy C-Means Clustering Algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.

[BGHW06] M. Benkert, J. Gudmundsson, F. Hübner et T. Wolle : Reporting flock patterns. Dans *Proceedings of the 14th European Symposium on Algorithms*, ESA, pages 660–671, 2006.

[BGHW08] M. Benkert, J. Gudmundsson, F. Hübner et T. Wolle : Reporting Flock Patterns. *Computational Geometry Theory and Applications*, 41(3): 111–125, novembre 2008.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider et B. Seeger : The R*-tree : An Efficient and Robust Access Method for Points and Rectangles. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[CBG10] D. R. Choffnes, F. E. Bustamante et Z. Ge : Crowdsourcing Service-level Network Event Monitoring. Dans *Proceedings of the ACM SIGCOMM Conference*, pages 387–398, 2010.

[CFNV04] M. Correia, N. Ferreira Neves et P. Veríssimo : How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems. Dans *Proceedings of the 23rd International Symposium on Reliable Distributed Systems*, SRDS, pages 174–183, 2004.

[Cho10] D. R. Choffnes : *Service-Level Network Event Detection from Edge Systems*. Thèse de doctorat, Northeastern University, 2010.

[Def77] D. Defays : An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[DLR77] A. P. Dempster, N. M. Laird et D. B. Rubin : Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society : Series B*, 39:1–38, 1977.

[EKSX96] M. Ester, H. P. Kriegel, J. Sander et X. Xu : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Dans *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 226–231, 1996.

[FSY05] A. Fiat, J. Saia et M. Young : Making Chord Robust to Byzantine Attacks. Dans *Proceedings of the 13rd European Symposium on Algorithms*, ESA, pages 803–814, 2005.

[HCDW12] C. Y. Hong, M. Caesar, N. Duffield et J. Wang : Tiresias : Online Anomaly Detection for Hierarchical Operational Network Data. Dans *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems*, ICDCS, pages 173–182, 2012.

[HE02] G. Hamerly et C. Elkan : Alternatives to the K-means Algorithm That Find Better Clusterings. Dans *Proceedings of the 11th International Conference on Information and Knowledge Management*, CIKM, pages 600–607, 2002.

[HW79] J. A. Hartigan et M. A. Wong : Algorithm AS 136 : A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[JLO07] C. S. Jensen, D. Lin et Beng-Chin O. : Continuous Clustering of Moving Objects. *Knowledge and Data Engineering, IEEE Transactions on*, 19(9): 1161–1174, 2007.

[JYZ+08] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen et H. T. Shen : Discovery of Convoys in Trajectory Databases. *Proceedings VLDB Endowment*, 1(1): 1068–1080, août 2008.

[KAD+07] R. Kotla, L. Alvisi, M. Dahlin, A. Clement et E. Wong : Zyzzyva : Speculative Byzantine Fault Tolerance. Dans *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP, pages 45–58, 2007.

[Kal60] R. E. Kalman : A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45, 1960.

[KR87] L. Kaufman et P. Rousseeuw : *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. 1987.

[LSW06] T. Locher, S. Schmid et R. Wattenhofer : eQuus : A Provably Robust and Locality-Aware Peer-to-Peer System. Dans *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, P2P, pages 3–11, 2006.

[MA04] S. Malgorzata et S. S. Adarshpal : A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[MCC03] M. L. Massie, B. N. Chun et D. E. Culler : The Ganglia Distributed Monitoring System : Design, Implementation And Experience. *Parallel Computing*, 30:2004, 2003.

[MM02] P. Maymounkov et D. Mazières : Kademlia : A Peer-to-Peer Information System Based on the XOR Metric. Dans *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS, pages 53–65, 2002.

[PM00] D. Pelleg et A. Moore : X-means : Extending K-means with Efficient Estimation of the Number of Clusters. Dans *Proceedings of the 17th International Conference on Machine Learning*, pages 727–734, 2000.

[PP06] K. Park et V. S. Pai : CoMon : A Mostly-scalable Monitoring System for PlanetLab. *SIGOPS Operating Systems Review*, 40(1):65–74, janvier 2006.

[RD01] A. Rowstron et P. Druschel : Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Dans *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Middleware, pages 329–350, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp et S. Shenker : A Scalable Content-addressable Network. Dans *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 161–172, 2001.

[RLH06] Y. Rekhter, T. Li et S. Hares : RFC 4271 : A Border Gateway Protocol 4 (BGP-4). Rapport technique, IETF, 2006.

[SEKX98] J. Sander, M. Ester, H.P. Kriegel et X. Xu : Density-Based Clustering in Spatial Databases : The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[Sib73] R. Sibson : SLINK : an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek et H. Balakrishnan : Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM Computer Communication Review*, 31(4):149–160, août 2001.

[VBT09] M. R. Vieira, P. Bakalov et V. J. Tsotras : On-line Discovery of Flock Patterns in Spatio-temporal Data. Dans *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS, pages 286–295, 2009.

[VRBV03] R. Van Renesse, K. P. Birman et W. Vogels : Astrolabe : A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions Computer Systems*, 21(2):164–206, mai 2003.

[YD04] P. Yalagandula et M. Dahlin : A Scalable Distributed Information Management System. Dans *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 379–390, 2004.

[YFG+12] H. Yan, A. Flavel, Z. Ge, A. Gerber, D. Massey, C. Papadopoulos, H. Shah et J. Yates : Argus : End-to-End Service Anomaly Detection and Localization From an ISP's Point of View. Dans *Proceedings of the IEEE INFOCOM Conference*, INFOCOM, pages 2756–2760, 2012.

[ZTG+09] Y. Zhao, Y. Tan, Z. Gong, X. Gu et M. Wamboldt : Self-correlating Predictive Information Tracking for Large-scale Production Systems. Dans *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC, pages 33–42, 2009.

Chapitre 3

Caractérisation de fautes

Nous définissons dans ce chapitre un modèle permettant de prendre en compte l'impact des fautes sur les perceptions des utilisateurs sans connaissance du réseau d'interconnexion. L'activation d'une faute dans le système peut induire une défaillance perçue par les utilisateurs. Cette défaillance est perçue au travers d'une variation anormale de la qualité d'un service consommé. Nous représentons un scénario possible de fautes impactant le système au travers d'une partition d'anomalies.

Cette modélisation va nous permettre de déterminer ce qui est discernable du point de vue des entités supervisées. Nous montrons qu'il est impossible de déterminer de manière certaine pour chaque entité ayant perçu une défaillance si cette défaillance est due à une faute locale (*faute isolée*) ou si cette défaillance est perçue par d'autres entités (*faute massive*).

En assouplissant le critère de faute (*faute isolée*, *faute massive* ou *état indéterminé*), il est possible de déterminer pour chaque variation de qualité anormale perçue, le type de faute à lui associer. De plus, nous montrons que la seule connaissance du voisinage, en termes de qualités, de chaque entité est suffisante pour déterminer pour chaque entité le type de faute ayant induit la défaillance perçue. Une connaissance plus importante, telle que celle qu'aurait un observateur global du système, n'apporte pas plus d'informations permettant de caractériser de manière certaine l'ensemble des fautes impactant le système.

3.1 Modèle

Cette section présente les notations et les concepts que nous utilisons pour la modélisation de l'impact des fautes sur les mesures de performances effectuées par les entités supervisées. Nous utilisons dans la suite de ce document les conventions suivantes :

— les variables sont notées en lettres minuscules : j, ℓ, p, q
— les ensembles sont notés en lettres majuscules : S, E
— les familles d'ensemble sont notées en lettres majuscules calligraphiées : $\mathcal{P}$
— les intervalles discrets $\{1, \ldots, n\}$ sont notés $[\![1, n]\!]$.

L'ensemble des notations utilisées dans cette section est résumé dans le tableau 3.1.

3.1.1 Préliminaires

On considère un ensemble $[\![1, n]\!]$ d'entités supervisées. Chacune de ces n entités consomme des services au travers du réseau choisis parmis d services notés $s_1, \ldots, s_d$. À chaque instant discret k, l'entité $j \in [\![1, n]\!]$ évalue une mesure de performance $q_{i,k}(j)$ du service s_i à valeurs dans $[0, 1]$. Si l'entité $j \in [\![1, n]\!]$ ne consomme pas le service s_i à l'instant k, $q_{i,k}(j)$ vaut 0. On modélise les mesures effectuées à l'instant k par l'entité j par un point $p_k(j) = (q_{1,k}(j), \ldots, q_{d,k}(j))$ de l'espace métrique $E = [0, 1]^d$. On parlera aussi d'espace des qualités pour désigner cet espace.

On modélise l'ensemble des mesures effectuées par les n entités par l'ensemble des points S_k de E, $S_k = (p_k(1), \ldots, p_k(n))$. Cet ensemble de points S_k est appelé *état du système* ou *configuration du système* à l'instant k. De plus, chaque entité est munie d'une fonction de détection de défaillance $a_k(j)$ qui retourne `VRAI` si une variation dans la mesure de performance d'un service consommé par l'entité j à l'instant k peut être considérée comme une défaillance. La notion de défaillance est très fortement dépendante du type de mesure effectué et du service consommé. Des méthodes (par exemple [Pag54, Hol04, Win60]) permettant cette détection de défaillance ont été évoquées auparavant et ne seront pas traitées ici.

Des fautes peuvent apparaître dans le système et leur impact peut être perçu par différentes entités. Nous nous intéressons à distinguer les fautes suivant le nombre d'entités supervisées impactées.

Définition 1 (fautes isolées / massives) *Étant donnés $k \geq 1$, S_{k-1}, S_k et $\tau \in [\![1, n-1]\!]$, une faute impactant strictement plus de τ entités supervisées est appelée faute massive. Dans le cas contraire, on parle de faute isolée.*

Les entités observant des variations de performances similaires sont susceptibles de percevoir les mêmes défaillances. On considère ici qu'une même faute a un impact similaire sur les mesures effectuées par les entités.

Nous traduisons cette intuition par l'hypothèse suivante : si les mesures effectuées par les entités avant la perception de la défaillance sont proches et qu'elles le sont encore après la perception de celle-ci, alors la défaillance est due à la même faute. Cette hypothèse est modélisée par une distance seuil entre les positions des entités dans l'espace des qualités : s'il existe une boule de rayon r contenant les positions de ces entités dans l'espace des qualités à l'instant $k-1$ et une autre contenant les positions de ces mêmes entités à l'instant k alors ces entités ont perçu la même défaillance. On appellera ce rayon r le rayon de cohérence.

3.1.2 Terminologie et notations employées

Afin de modéliser de l'impact des fautes sur le système, nous introduisons les notions que nous utiliserons tout au long de ce chapitre. Tout d'abord, dans un soucis de simplicité, nous utilisons la norme infinie $\|\cdot\|$ définie pour tout $x = (x_1, \ldots, x_d) \in E$

par $\|x\| = \max\{x_1, \ldots, x_d\}$. D'autre part, la distance entre deux points $x, y \in E$ est définie par la norme du vecteur $\|x-y\|$. L'espace E étant de dimension finie d toutes les normes sont équivalentes et donc les résultats fournis restent valables à des constantes près.

Définition 2 (Ensemble r-cohérent) *Pour tout $r \in [0, 1/4)$, un sous-ensemble $B \subseteq [\![1, n]\!]$ est dit* r-cohérent *à l'instant $k \geq 0$ si la distance maximale entre tous points $i, j \in B$ est inférieure ou égale à $2r$:*

$$\forall (i, j) \in B^2, \|p_k(i) - p_k(j)\| \leq 2r.$$

Propriété 1 *Pour tout $k \geq 0$, pour tout $r \in [0, 1/4)$ et pour tout $B \subseteq [\![1, n]\!]$, on a l'équivalence suivante :*

$$\exists o \in E, \forall j \in B, \|p_k(j) - p_k(o)\| \leq r \iff B \text{ est } r\text{-cohérent à l'instant } k.$$

Preuve ($\Rightarrow$) Soient $B \subseteq [\![1, n]\!]$ et $o \in E$ tels que pour tout $j \in B$, nous ayons $\|p_k(j) - p_k(o)\| \leq r$. Cela implique que tous les éléments de B sont contenus dans une boule de rayon r centrée en o. On a donc $\forall (i, j) \in B^2, \|p_k(i) - p_k(j)\| \leq 2r$. Par conséquent, B est un ensemble r-cohérent.

($\Leftarrow$) Soit $B = [\![1, \ell]\!] \subseteq [\![1, n]\!]$ un ensemble r-cohérent à l'instant k. D'après la définition 2 cela signifie que $\forall (i, j) \in B^2, \|p_k(i) - p_k(j)\| \leq 2r$. De plus, d'après la définition de la norme employée, la distance entre deux points $(x, y) \in E^2$ est définie par $\|x - y\| = \max\{|x_1 - y_1|, \ldots, |x_d - y_d|\}$. On a donc $\forall (i, j) \in B^2, \forall m \in [\![1, d]\!], |q_{m,k}(i) - q_{m,k}(j)| \leq 2r$. Soit le point $o = (o_1, \ldots, o_d) \in E$. On considère la m-ème coordonnée $q_{m,k}(j)$ de chaque point $j \in B$. Ces coordonnées sont toutes incluses dans le segment $[\min\{q_{m,k}(1), \ldots, q_{m,k}(\ell)\}, \max\{q_{m,k}(1), \ldots, q_{m,k}(\ell)\}]$. On définit alors o_m comme le milieu de ce segment. On a :

$$\forall m \in [\![1, d]\!], o_m = \frac{\max\{q_{m,k}(1), \ldots, q_{m,k}(\ell)\} - \min\{q_{m,k}(1), \ldots, q_{m,k}(\ell)\}}{2}.$$

L'ensemble B étant r-cohérent à l'instant k, on a $\forall (i, j) \in B^2, |q_{m,k}(i) - q_{m,k}(j)| \leq 2r$. Par conséquent, on a $\forall j \in B, \forall m \in [\![1, d]\!], |o_m - q_{m,k}(j)| \leq r$ et donc $\forall j \in B, \|p_k(j) - p_k(o)\| \leq r$. □

Définition 3 (Ensemble r-cohérent maximal) *Pour tout $r \in [0, 1/4)$, un sous-ensemble $B \subseteq [\![1, n]\!]$ est dit* maximal r-cohérent *à l'instant k si les deux conditions suivantes sont vérifiées :*

- *B est un ensemble r-cohérent à l'instant k,*
- *$\forall j \in [\![1, n]\!] \setminus B$, $B \cup \{j\}$ n'est pas un ensemble r-cohérent à l'instant k.*

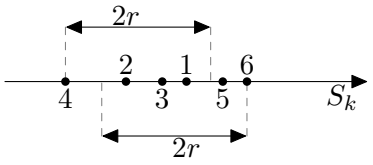


FIGURE 3.1 – Les ensembles maximaux r-cohérents $B_1 = \{1, 2, 3, 4\}$ et $B_2 = \{1, 2, 3, 5, 6\}$ contenant le point 1. Tout sous-ensemble de B_1 ou B_2 est aussi r-cohérent.

Exemple 6 *La figure 3.1 illustre ces deux notions. On considère un ensemble de* 6 *entités labellisées* $\{1, 2, 3, 4, 5, 6\}$ *à l'instant* k*. Ces entités consomment un unique service (donc* $d = 1$*) et mesurent la performance de celui-ci. On place ces entités dans l'espace des qualité* E *qui se réduit alors au segment* $[0, 1]$ *et les points correspondent à la position de chaque entité supervisée dans* E*. La distance entre tout couple d'éléments de* $B_1 = \{1, 2, 3, 4\}$ *ou tout couple d'éléments de* $B_2 = \{1, 2, 3, 5, 6\}$ *est inférieure à* $2r$*. Ces ensembles sont donc* r*-cohérents. De plus, ces ensembles* B_1 *et* B_2 *sont maximaux : en effet, tout ajout à* B_1 *d'un élément de* B_2 *augmente la distance maximale entre les éléments de* B_1 *au delà de* $2r$ *brisant ainsi la condition de* r*-cohérence. Le même raisonnement s'applique à* B_2*.*

Ces deux notions nous permettent d'introduire la notion de mouvement permettant de capturer les variations de perception des entités fortement corrélées.

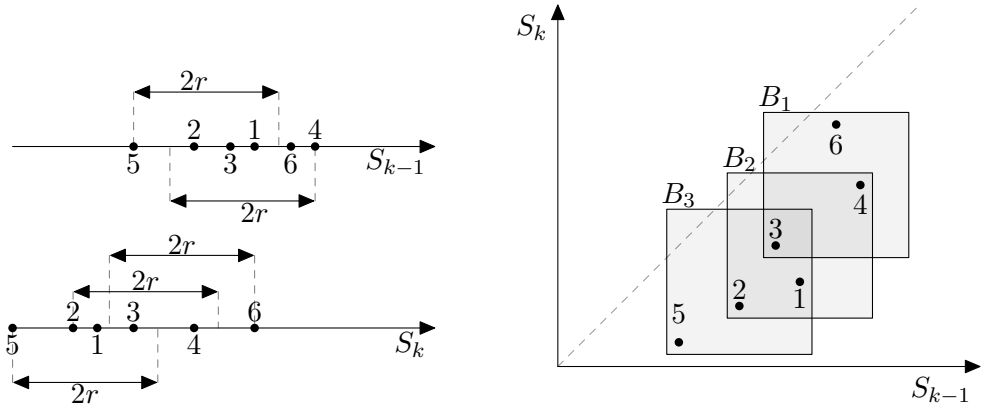
Définition 4 (Mouvement r-cohérent) *Pour tout* $k \geq 1$*, pour tout* $r \in [0, 1/4)$*, un sous-ensemble* $B \subseteq [\![1, n]\!]$ *a un mouvement* r*-cohérent dans l'intervalle de temps* $[k-1, k]$ *si* B *est un ensemble* r*-cohérent aux instants* $k-1$ *et* k*.*

Définition 5 (Mouvement r-cohérent maximal) *Pour tout* $k \geq 1$*, pour tout* $r \in [0, 1/4)$*, un sous-ensemble* $B \subseteq [\![1, n]\!]$ *a un mouvement* r*-cohérent maximal dans l'intervalle de temps* $[k-1, k]$ *si les deux conditions suivantes sont vérifiées :*

- B *a un mouvement* r*-cohérent dans l'intervalle de temps* $[k-1, k]$*,*
- $\forall j \in [\![1, n]\!] \setminus B, B \cup \{j\}$ *n'a pas de mouvement* r*-cohérent dans l'intervalle de temps* $[k-1, k]$*.*

Remarque 1 *Si un sous-ensemble* $B \subseteq [\![1, n]\!]$ *a un mouvement* r*-cohérent dans l'intervalle de temps* $[k-1, k]$*, alors soit ce mouvement est maximal, soit il existe* $B' \subseteq [\![1, n]\!], B \subseteq B'$ *tel que* B' *ait un mouvement maximal* r*-cohérent.*

Exemple 7 *La figure 3.2 illustre la notion de mouvements* r*-cohérents. À gauche, sur la figure 3.2(a), sont représentées les positions de six entités* $\{1, 2, 3, 4, 5, 6\}$ *consommant un unique service aux instants* $k-1$ *et* k *ainsi que les ensembles* r*-cohérents maximaux contenant chacune d'elles. La figure 3.2(b) représente ces mêmes entités dans l'espace des trajectoires* $T = E \times E$*. La variation de qualité perçue par l'entité* j *est représentée par le point* $(q_{1,k-1}(j), \ldots, q_{d,k-1}(j), q_{1,k}(j), \ldots, q_{d,k}(j))$*.*



(a) Deux ensembles r-cohérents maximaux à l'instant $k-1$: $\{1,2,3,5\}$ et $\{1,2,3,4,6\}$. On a trois ensembles r-cohérents maximaux à l'instant k : $\{1,2,3,5\}, \{1,2,3,4\}, \{3,4,6\}$.

(b) Trois ensembles ayant des mouvements r-cohérents maximaux : $B_1 = \{1,2,3,5\}$, $B_2 = \{1,2,3,4\}, B_3 = \{3,4,6\}$.

FIGURE 3.2 – Représentation des variations de qualité dans l'espace des trajectoires.

La droite $y = x$ est tracée en pointillés, les points situés sur cette droite traduisent l'absence de variation de qualité mesurée par les entités dans l'intervalle de temps $[k-1,k]$: pour chacune d'elles on a $p_k(j) = p_{k-1}(j)$. Les points situés au dessus de cette droite traduisent une amélioration de la qualité mesurée par les entités dans l'intervalle de temps $[k-1,k]$ tandis que ceux situés en dessous traduisent une dégradation de la qualité perçue par les entités dans l'intervalle de temps $[k-1,k]$. Ici, les six entités ont perçu une dégradation de leur qualité.

Les mouvements r-cohérents correspondent aux points contenus dans le produit cartésien d'une boule de rayon r à l'instant $k-1$ avec une autre boule de rayon r à l'instant k. Ces mouvements sont représentés par les carrés grisés B_1, B_2 et B_3. Nous utiliserons cette représentation dans le reste de ce document.

Finalement, nous classifions les mouvements r-cohérents suivant le nombre d'entités supervisées qui les composent. Cette classification est au centre de la distinction que nous établissons entre les fautes isolées et les fautes massives.

Définition 6 (Mouvement τ-dense) *Pour tout $r \in [0, 1/4)$, pour tout $\tau \in [\![1, n-1]\!]$, et pour tout sous-ensemble $B \subseteq [\![1, n]\!]$ ayant un mouvement r-cohérent dans l'intervalle de temps $[k-1,k]$, on dit que B a un mouvement r-cohérent τ-dense dans l'intervalle de temps $[k-1,k]$ si $|B| > \tau$, sinon on dit que B a un mouvement r-cohérent non τ-dense dans cet intervalle.*

Dans la suite, on utilisera "mouvement τ-dense" (respectivement "mouvement non τ-dense") au lieu de "mouvement r-cohérent τ-dense" (respectivement " mouvement r-cohérent non τ-dense") lorsque le contexte est clair.

3.1.3 Modélisation de l'impact des fautes

Chaque entité supervisée j consomme un sous-ensemble des d services. Pour chaque service consommé, l'entité j effectue localement une mesure de performance de bout-en-bout. Ces mesures de performances sont utilisées en entrée de la fonction de détection de défaillance $a_k(j)$. Si une variation de performance est considérée comme une défaillance par la fonction de détection de défaillance $a_k(j)$, celle-ci retourne `VRAI`. On définit la trajectoire anormale d'une entité du système dans l'espace des qualités comme la conséquence sur les mesures de performance d'une faute dans le système.

Définition 7 (Trajectoire anormale) *Une entité $j \in [\![1, n]\!]$ a une trajectoire anormale dans l'intervalle de temps $[k-1, k]$ si $a_k(j) =$ `VRAI`. L'ensemble des entités ayant une trajectoire anormale dans l'intervalle de temps $[k-1, k]$ est noté A_k. Formellement, on a :*

$$A_k = \{j \in [\![1, n]\!] \mid a_k(j) = \texttt{VRAI}\}\,.$$

Nous cherchons à déterminer pour chaque entité percevant une défaillance dans l'intervalle de temps $[k-1, k]$ si la faute qui a causé cette défaillance a impacté un grand nombre d'entités ou seulement quelques unes. Pour cela, nous déterminons pour chaque entité la présence d'autres entités dans son voisinage dans l'espace des qualités aux instants $k-1$ et k. Par exemple, si plus de τ entités ayant des trajectoires anormales restent proches les unes des autres, nous considérons que ces trajectoires anormales sont dues à une même faute et donc que ces entités perçoivent une défaillance due à une faute massive. Nous pouvons alors restreindre la recherche des causes de cette défaillance à ces seules entités impactées. Cependant, si l'intervalle de temps entre les instants de mesure est trop important, il est possible que ces mêmes entités perçoivent différentes défaillances, rendant impossible toute reconstruction de scénario d'explication. Nous imposons donc trois restrictions nécessaires à la reconstruction de ces scénarios :

R1 : Dans l'intervalle de temps $[k-1, k]$, la trajectoire anormale de l'entité supervisée $j \in A_k$ n'est due qu'à une unique faute.

R2 : Une faute a un impact similaire sur toutes les entités supervisées qui la perçoivent. En particulier, si un ensemble B d'entités supervisées forme un ensemble r-cohérent avant la perception de la défaillance (*i.e.* à l'instant $k-1$), alors celles-ci forment encore un ensemble r-cohérent à l'instant k et donc par la définition 4, cet ensemble a un mouvement r-cohérent dans l'intervalle de temps $[k-1, k]$.

R3 : Si une faute impacte strictement plus de τ entités alors pour chacune d'elles il existe un ensemble ayant un mouvement τ-dense. Celui-ci ne contient pas nécessairement l'ensemble des entités impactées par cette faute massive.

Réciproquement, dans un ensemble ayant un mouvement τ-dense, il existe au moins une entité ayant été effectivement impactée par une faute massive.

Ces restrictions imposent que les entités percevant une défaillance due à une faute massive aient un mouvement r-cohérent. Cependant, une faute peut engendrer plusieurs

mouvements r-cohérents à des positions différentes dans l'espace des qualités. Ces restrictions sont formalisées par le partitionnement de l'ensemble A_k en sous-ensembles ayant des mouvements r-cohérents. Ce partitionnement est tel que :

(i) tous les mouvements r-cohérents non τ-denses sont suffisamment éloignés les uns des autres, de sorte qu'il soit impossible de reconstituer un mouvement τ-dense à partir de ces éléments. En d'autres termes, la probabilité que plus de τ fautes indépendantes impactent des entités supervisées dont les mesures effectuées à deux instants consécutifs sont similaires est négligeable.

Cette condition traduit le fait qu'il est impossible de distinguer une faute impactant un ensemble d'entités supervisées d'un ensemble de fautes impactant chacune une entité supervisée de manière similaire aux autres fautes.

Par exemple, des passerelles de connexion utilisateurs subissant chacune une faute isolée impactant la connexion réseau percevraient les mêmes variations de mesure de performance qu'un ensemble de passerelles de connexion utilisateurs percevant une même défaillance réseau.

(ii) tout mouvement r-cohérents non τ-dense est suffisamment éloigné d'un mouvement r-cohérent τ-dense, de sorte qu'il soit impossible de les regrouper au sein d'un unique ensemble ayant un mouvement r-cohérent τ-dense. En d'autres termes, la probabilité que plusieurs fautes indépendantes impactent plus de τ entités supervisées dont les mesures effectuées sont similaires est négligeable.

Cette condition traduit le fait qu'il est impossible de distinguer une faute impactant un ensemble d'entités supervisées, de deux fautes impactant l'une un faible nombre (inférieur à τ) d'entités et la seconde le reste des entités. Considérons par exemple d'une part une passerelle utilisateur sujette à une faute isolée et d'autre part un ensemble de passerelles de connexion utilisateurs percevant une même défaillance réseau, et supposons que toutes ces entités perçoivent une même variation de qualité. Dans ce cas, l'entité sujette à une faute isolée exhibe alors une trajectoire anormale similaire à celles des autres entités. Il est donc impossible de discerner l'entité impactée par la faute isolée des autres entités percevant une faute massive.

Afin de prendre en compte ces restrictions, nous partitionnons l'ensemble A_k des entités de $[\![1, n]\!]$ ayant une trajectoires anormales dans l'intervalle de temps $[k-1, k]$ en sous-ensembles appelés anomalies. Les anomalies ont des mouvements r-cohérents dans l'espace des qualités. Les entités d'une anomalie perçoivent donc une défaillance de manière cohérente.

Une partition d'anomalies $\mathcal{P}_k$ représente un scénario possible de fautes, respectant les restrictions **R1**, **R2** et **R3**, qui impactent le système. On dit que la partition d'anomalies a fait évoluer le système de l'état S_{k-1} à l'état S_k. Ce partitionnement de A_k est formellement défini comme suit.

Définition 8 (Partition d'anomalies $\mathcal{P}_k$) *Pour tout $k \geq 1$, pour tout $\tau \in [\![1, n-1]\!]$ et pour tout $r \in [0, 1/4)$, la partition $\mathcal{P}_k$ de A_k est une partition d'anomalies à l'instant k si elle est constituée d'ensembles $C_1, \ldots, C_\ell$, non vides et deux à deux disjoints, ayant*

des mouvements r-cohérents et vérifiant les conditions C1, C2 *suivantes. Les ensembles* $C_1, \ldots, C_\ell$ *sont appelés anomalies.*

C1 : $\forall B \subseteq \bigcup_{|C_i| \leq \tau} C_i$, *B a un mouvement r-cohérent non* τ*-dense ou B n'a pas de mouvement r-cohérent.*

C2 : $\forall i \in [\![1, \ell]\!]$, C_i *a un mouvement r-cohérent* τ*-dense* $\Rightarrow \forall B \subseteq \bigcup_{|C_j| \leq \tau} C_j, B \cup C_i$ *n'a pas un mouvement r-cohérent.*

Par extension, pour tout élément $j \in A_k$ on notera $\mathcal{P}_k(j)$ l'unique partie de $\mathcal{P}_k$ contenant l'élément j.

Finalement on distingue les anomalies isolées des massives suivant le cardinal des parties $C_1, \ldots, C_\ell$ de $\mathcal{P}_k$.

Définition 9 (Anomalies Massives / Isolées) *Soit* $\mathcal{P}_k$ *une partition d'anomalies. Un élément* $C \in \mathcal{P}_k$ *est appelé anomalie massive de* $\mathcal{P}_k$ *dans l'intervalle de temps* $[k-1, k]$ *si* $|C| > \tau$*. Dans le cas contraire, on l'appelle anomalie isolée de* $\mathcal{P}_k$*.*

L'ensemble des entités supervisées impactées par une anomalie massive de $\mathcal{P}_k$ *dans l'intervalle de temps* $[k-1, k]$ *est noté* $M_{\mathcal{P}_k}$*. De la même manière on note* $I_{\mathcal{P}_k}$*, l'ensemble des entités supervisées impactées par une anomalie isolées de* $\mathcal{P}_k$ *dans l'intervalle de temps* $[k-1, k]$*.*

Formellement on a :

$$M_{\mathcal{P}_k} = \{j \in A_k \mid |\mathcal{P}_k(j)| > \tau\},$$

$$I_{\mathcal{P}_k} = \{j \in A_k \mid |\mathcal{P}_k(j)| \leq \tau\}.$$

Ces ensembles sont complémentaires dans A_k et induisent une décomposition naturelle de A_k. On a :

$$A_k = M_{\mathcal{P}_k} \cup I_{\mathcal{P}_k} \text{ et } M_{\mathcal{P}_k} \cap I_{\mathcal{P}_k} = \emptyset. \tag{3.1}$$

Bien que la définition de partition d'anomalie puisse sembler complexe, le lemme 1 montre qu'on peut toujours construire une partition d'anomalies, et qu'en général celle-ci n'est pas unique.

Lemme 1 *Pour tout* $k \geq 1$*, pour tout* $A_k \neq \emptyset$*, pour tout* $\tau \in [\![1, n-1]\!]$*, pour tout* $r \in [0, 1/4)$ *et pour tous états consécutifs* S_{k-1}, S_k *du système, il existe toujours une partition* $\mathcal{P}_k$ *de* A_k *qui est une partition d'anomalies. Cette partition n'est en général pas unique.*

Preuve On montre dans un premier temps l'existence de partitions d'anomalies, puis on illustrera sur un exemple que celles ci ne sont pas nécessairement uniques.

(Existence) L'algorithme 1 décrit une manière simple de construire une partition d'anomalies $\mathcal{P}_k$ de A_k. On initialise l'ensemble des entités à traiter S avec les éléments de A_k et le résultat $\mathcal{P}_k$ avec l'ensemble vide. Tant qu'il reste des éléments à traiter (*i.e.* S n'est pas vide), on choisit aléatoirement un élément j dans S. On choisit alors un sous-ensemble C de S tel que j appartienne à ce sous-ensemble C et que C ait

Algorithme 1 : Construction d'une partition d'anomalie de A_k.

Données : $S_{k-1}, S_k, \tau \in [\![1, n-1]\!], r \in \left[0, \frac{1}{4}\right)$.
Entrées : A_k
Sorties : Une partition d'anomalie

```
début
    S ← A_k;
    P_k ← {};
    tant que S ≠ ∅ faire
        Choisir j ∈ S;
        Choisir C ⊆ S tel que j ∈ C, C ∉ P_k et C a un mouvement
        r-consistent maximal dans S;
        S ← S \ C ;
        P_k ← P_k ∪ {C};
    fin
    retourner P_k;
fin
```

un mouvement maximal r-cohérent parmi les éléments de S. Cet ensemble C est alors ajouté à $\mathcal{P}_k$ et ses éléments sont retirés de S. À chaque itération, on supprime donc des éléments de S. La taille de S est donc strictement décroissante, l'algorithme termine.

Montrons à présent par récurrence qu'à chaque itération $\mathcal{P}_k$ satisfait les conditions C1 et C2 de la définition 8. Le premier élément ajouté à $\mathcal{P}_k$ a un mouvement r-cohérent maximal par définition. Comme il s'agit du premier élément, il n'existe pas d'autre partie de $\mathcal{P}_k$ permettant éventuellement de lui adjoindre des éléments.

Supposons à présent, qu'à la fin de la ℓ-ième itération, $\mathcal{P}_k = \{C_1, \ldots, C_\ell\}$ satisfait les conditions C1 et C2 de la définition 8. Montrons alors qu'à la fin de l'itération suivante, $\mathcal{P}_k$ les satisfait encore. On choisit un élément j de S. On choisit ensuite $C \subseteq S$ tel que j appartienne à C et que C ait un mouvement maximal r-cohérent parmi les éléments de S. Par construction, $\forall i \in [\![1, \ell]\!], C_i \in \mathcal{P}_k$ a un mouvement r-cohérent maximal parmi les éléments restants $S \setminus \cup_{1 \leq i \leq \ell} C_i$. Donc, d'après la définition 5, $\forall j \in S \setminus \cup_{1 \leq i \leq \ell} C_i, C_i \cup \{j\}$ n'a pas un mouvement r-cohérent. En particulier, $\forall j \in C, \forall i \in [\![1, \ell]\!], C_i \cup \{j\}$ n'a pas un mouvement r-cohérent et donc les conditions C1 et C2 sont satisfaites par $\mathcal{P}_k$. Par hypothèse de récurrence, $\mathcal{P}_k$ satisfait donc C1 et C2 à chaque itération.

Enfin, A_k n'est pas vide et donc pour tout élément $j \in A_k$, il existe une unique partie $C \in \mathcal{P}_k$ contenant j. À la fin de l'exécution de l'algorithme, tous les éléments de A_k appartiennent à une partie de $\mathcal{P}_k$ et donc $\mathcal{P}_k$ est bien une partition de A_k.

(Non unicité) Étant donnés A_k, S_{k-1} and S_k, on illustre la possibilité de construire diverses partitions d'anomalies. Considérons l'exemple de la figure 3.3. Celle-ci illustre les mesures de performances prises par dix entités supervisées à l'instant k en fonction de celles mesurées à l'instant $k-1$, ainsi que les mouvements r-cohérents maximaux possibles. Supposons qu'on a $\tau = 3$ et que toutes les entités supervisées ont une trajectoire anormale dans l'intervalle de temps $[k-1, k]$.

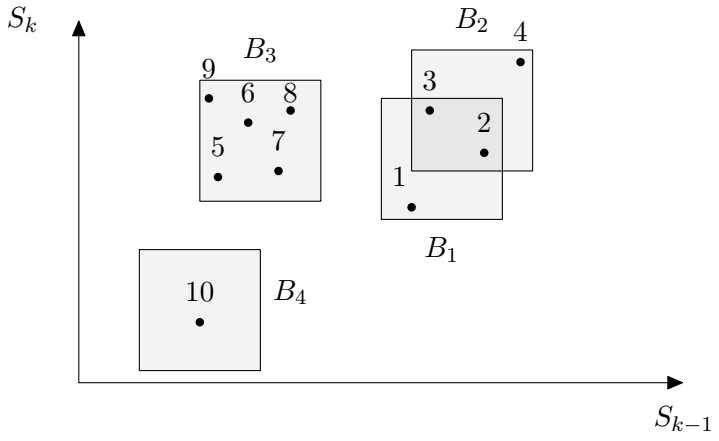


FIGURE 3.3 – Mesures de performances de dix entités supervisées à l'instant k en fonction de leurs mesures à l'instant $k-1$. Les quatre mouvements r-cohérents maximaux C_1, C_2, C_3, C_4 sont illustrés par les carrés gris. Le seuil de densité τ vaut 3.

Appliquons à présent l'algorithme 1 et supposons que l'entité 1 soit choisie initialement. On sélectionne ensuite l'ensemble $C = \{1, 2, 3\}$. L'algorithme s'exécute ensuite de manière déterministe et construit la partition d'anomalies suivante $\mathcal{P}'_k = \{\{1, 2, 3\}, \{4\}, \{5, 6, 7, 8, 9\}, \{10\}\}$. Cependant, si on choisit initialement l'entité 4, on ajoute à la partition courante l'ensemble $\{2, 3, 4\}$. L'algorithme construit la partition $\mathcal{P}''_k = \{\{1\}, \{2, 3, 4\}, \{5, 6, 7, 8, 9\}, \{10\}\}$. D'après la première partie de la preuve, ces deux partitions sont des partitions d'anomalies. Étant donnés A_k, S_{k-1} and S_k, les partitions d'anomalies ne sont donc pas nécessairement uniques. □

Une partition d'anomalies $\mathcal{P}_k$ représente un scénario possible de fautes respectant les restrictions **R1**, **R2** et **R3** qui impactent le système. Une anomalie représente une défaillance due à une faute potentielle et perçue par un ensemble d'entités. Dans ce contexte, on note $\mathcal{R}_k$ la partition d'anomalies correspondant au scénario réel de fautes ayant impacté le système, et $I_{\mathcal{R}_k}$ l'ensemble des entités supervisées ayant effectivement subi une faute isolée et $M_{\mathcal{R}_k}$ celles ayant effectivement perçu une faute massive.

3.2 Problèmes étudiés

Nous cherchons à discerner les entités qui ont effectivement perçu une défaillance due à une faute isolée de celles qui ont perçu une défaillance due à une faute massive sans connaître la réalité des faits. Nous cherchons donc à déterminer les entités de $I_{\mathcal{R}_k}$ et de $M_{\mathcal{R}_k}$ sans connaître $\mathcal{R}_k$.

Considérons un observateur qui serait capable à chaque instant k de connaître l'état du système S_k dans son intégralité. De plus, pour chaque entité supervisée j, cet observateur a accès au résultat de la fonction $a_k(j)$. Il connaît donc l'ensemble A_k. En revanche, cet observateur ne connaît pas la partition $\mathcal{R}_k$.

TABLE 3.1 – Liste des notations

Notation	Signification
E	Espace des qualités (Section 3.1.1)
$p_k(j)$	Position dans E à l'instant k de l'entité j (Section 3.1.2)
r	Rayon de cohérence (Section 3.1.1)
τ	Seuil de densité (Définition 6)
$a_k(j)$	Détection d'anomalie par l'entité j à l'instant k (Définition 7)
A_k	Ensemble des entités percevant une défaillance dans l'intervalle de temps $[k-1,k]$ (Relation 7)
S_k	État ou configuration du système à l'instant k (Section 3.1.1)
$\mathcal{P}_k$	Partition d'anomalies à l'instant k (Définition 8)
$\mathcal{R}_k$	Scénario réel de fautes ayant impacté le système dans l'intervalle de temps $[k-1,k]$
$M_{\mathcal{P}_k}$	Ensemble des entités impactées par une faute massive dans l'intervalle de temps $[k-1,k]$ dans le scénario $\mathcal{P}_k$ (Définition 9)
$I_{\mathcal{P}_k}$	Ensemble des entités impactées par une faute isolée dans l'intervalle de temps $[k-1,k]$ dans le scénario $\mathcal{P}_k$ (Définition 9)

Cet observateur cherche à séparer les entités de A_k en deux sous-ensembles I_k et M_k tels que I_k contienne toutes les entités pour lesquelles l'observateur suppose qu'elles ont perçu une faute isolée et M_k contienne toutes les entités pour lesquelles l'observateur suppose qu'elles ont perçu une faute massive sans connaître $\mathcal{R}_k$.

On défini alors le problème suivant :

Problème 1 (Caractérisation de fautes) *Étant donnés $k \geq 1$, S_k, A_k, $r \in [0, 1/4)$ et $\tau \in [\![1, n-1]\!]$, un observateur global est-il toujours capable de construire M_k et I_k de sorte que $M_k = M_{\mathcal{R}_k}$ et $I_k = I_{\mathcal{R}_k}$ sans connaître $\mathcal{R}_k$?*

On dit que ce problème peut être résolu si un observateur global peut reconstruire les ensembles $M_{\mathcal{R}_k}$ et $I_{\mathcal{R}_k}$ de manière systématique.

Théorème 1 (Impossibilité) *Étant donnés $k \geq 1$, S_k, A_k, $r \in [0, 1/4)$ et $\tau \in [\![1, n-1]\!]$, il est impossible pour un observateur global de construire M_k et I_k de manière systématique de sorte que $M_k = M_{\mathcal{R}_k}$ et $I_k = I_{\mathcal{R}_k}$ sans connaître $\mathcal{R}_k$.*

Preuve On montre par un contre exemple que le problème de la caractérisation de fautes ne peut pas être résolu.

Considérons le scénario illustré par la figure 3.4. On considère un ensemble de cinq entités $S = \{1, 2, 3, 4, 5\}$, chacune subissant des variations de mesures de performances dans l'intervalle de temps $[k-1, k]$. On suppose de plus que le seuil de densité τ vaut 3 et que toutes les entités ont des trajectoires anormales (*i.e.* $\forall j \in S, a_k(j) = \texttt{VRAI}$). Deux ensembles $B_1 = \{1, 2, 3, 4\}$ et $B_2 = \{2, 3, 4, 5\}$ ayant un mouvement maximal sont représentés par les carré grisés.

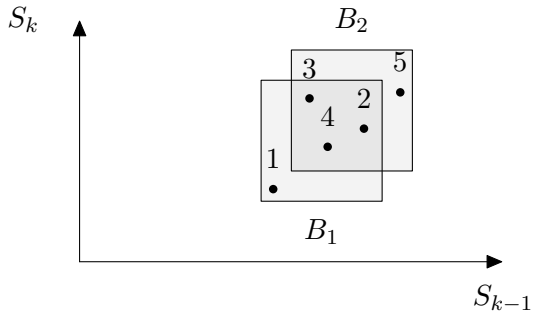


FIGURE 3.4 – Un scénario simple d'indécision.

Soient $\mathcal{P}_k^1 = \{\{1,2,3,4\},\{5\}\}$ et $\mathcal{P}_k^2 = \{\{1\},\{2,3,4,5\}\}$ deux partitions d'anomalies. On a $\tau = 3$, par conséquent $M_{\mathcal{P}_k^1} = \{1,2,3,4\}$ et $I_{\mathcal{P}_k^1} = \{5\}$. De la même manière, on a $M_{\mathcal{P}_k^2} = \{2,3,4,5\}$ et $I_{\mathcal{P}_k^2} = \{1\}$.

Bien qu'un observateur global soit capable de reconstruire ces deux partitions d'anomalies, il est incapable de déterminer laquelle des deux correspond au scénario réel. Il est donc incapable de déterminer de manière certaine les ensembles à construire. Le problème 1 est donc impossible à résoudre. □

Nous venons de montrer l'existence de configurations du système ne permettant pas à un observateur global de distinguer avec certitude les entités supervisées ayant été impactées par des fautes isolées de celles impactées par des fautes massives.

On s'intéresse alors à affaiblir le problème 1 en partitionnant l'ensemble A_k des entités supervisées percevant une défaillance en trois sous-ensembles I_k, M_k, U_k, tels que I_k et M_k contiennent les entités pour lesquelles l'observateur global peut déterminer avec certitude qu'elles ont été impactées par une faute isolée (I_k) ou globale (M_k). Ces ensembles sont formellement définis ainsi :

$$I_k = \{\ell \in A_k \mid \forall \mathcal{P}_k, |\mathcal{P}_k(\ell)| \leq \tau\} \quad , \quad M_k = \{\ell \in A_k \mid \forall \mathcal{P}_k, |\mathcal{P}_k(\ell)| > \tau\} \tag{3.2}$$

avec $\mathcal{P}_k(\ell)$ correspondant à la partie de $\mathcal{P}_k$ contenant ℓ.

Cela signifie donc que quelle que soit la partition d'anomalie $\mathcal{P}_k$, on a $M_k \subset M_{\mathcal{P}_k}$ et $I_k \subset I_{\mathcal{P}_k}$ et donc en, particulier, on a $M_k \subset M_{\mathcal{R}_k}$, $I_k \subset I_{\mathcal{R}_k}$. L'ensemble contenant les entités pour lesquelles un observateur global est incapable de déterminer avec certitude le type de faute perçue par celles-ci est noté U_k. Cet ensemble est formellement défini comme suit :

Définition 10 (État indéterminé) *Une entité $j \in A_k$ est dans un état indéterminé s'il existe deux partitions d'anomalies $\mathcal{P}_k$ and $\mathcal{P}'_k$ telles que $j \in I_{\mathcal{P}_k}$ et $j \in M_{\mathcal{P}'_k}$. L'ensemble des entités en état indéterminé dans l'intervalle de temps $[k-1,k]$ est noté U_k.*

Corollaire 1 *Pour tout instant $k \geq 1$, pour tout S_k, pour tout A_k, pour tout $r \in [0, 1/4)$ et pour tout $\tau \in [\![1, n-1]\!]$, on a*

$$U_k = \emptyset \Longrightarrow \textit{ la problème 1 peut être résolu de manière systématique.}$$

Preuve Supposons $U_k = \emptyset$. D'après la définition 10, cela signifie que toute entité de A_k appartient soit à M_k soit à I_k. Supposons $j \in M_k$, le même raisonnement s'appliquant pour $j \in I_k$. D'après la relation (3.2), $j \in M_k \Leftrightarrow \forall \mathcal{P}_k, |\mathcal{P}_k(j)| > \tau$. Cela signifie donc en particulier que pour toute partition d'anomalie $\mathcal{P}_k, M_{\mathcal{P}_k} = M_{\mathcal{R}_k}$. Ainsi, il est suffit d'appliquer l'algorithme 1 afin de construire une partition $\mathcal{P}_k$ et donc d'en déduire $M_{\mathcal{P}_k} = M_{\mathcal{R}_k}$. □

On définit à présent, une version affaiblie du problème 1.

Problème 2 (Caractérisation faible de fautes) *Étant donnés $k \geq 1$, S_k et A_k, $r \in [0, 1/4)$ et $\tau \in [\![1, n-1]\!]$, un observateur global est-il toujours capable de construire M_k, I_k et U_k de sorte que $M_k \subseteq M_{\mathcal{R}_k}$, $I_k \subseteq I_{\mathcal{R}_k}$ et $M_k \cup I_k \cup U_k = A_k$ sans connaître $\mathcal{R}_k$?*

L'introduction de U_k nous permet à présent de résoudre ce problème de manière déterministe. La section suivante détaille des conditions nécessaires et suffisantes pour l'appartenance d'une entité supervisée à chacun de ces trois ensembles.

3.3 Conditions d'appartenance à I_k, M_k ou U_k

Nous détaillons dans cette section comment, pour chaque entité supervisée percevant une défaillance, déterminer si celle-ci appartient à M_k, I_k ou U_k. Une approche naïve consisterait à générer toutes les partitions d'anomalies possibles faisant évoluer le système de S_{k-1} à S_k.

Cependant, ce type d'approche n'est pas envisageable en pratique. En effet, le nombre de partitions (classiques) d'un ensemble à n éléments est donné par le nombre de Bell B_n. La suite $(B_n)_{n\geq 1}$ croît exponentiellement avec n, comme le montre le tableau 3.2. Il n'est donc pas raisonnable d'envisager cette approche pour des systèmes large-échelle.

TABLE 3.2 – Nombre B_n de partitions d'un ensemble de n éléments.

n	1	2	3	4	5
B_n	1	2	5	15	52
n	6	7	8	9	10
B_n	203	877	4140	21147	115975
n	11	12	13	14	15
B_n	678570	4213597	27644437	190899322	1382958545
n	16	17	18	19	20
B_n	10480142147	82864869804	682076806159	5832742205057	51724158235372

TABLE 3.3 – Liste des notations

Notation	Signification
M_k	Ensemble des entités impactées par une faute massive dans l'intervalle de temps $[k-1,k]$ (Section 3.2)
I_k	Ensemble des entités impactées par une faute isolée dans l'intervalle de temps $[k-1,k]$ (Section 3.2)
U_k	Ensemble des entités en état indéterminé dans l'intervalle de temps $[k-1,k]$ (Définition 10)
$\mathcal{M}_k(j)$	Famille des ensembles contenant l'entité j ayant un mouvement r-cohérent dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$\mathcal{W}_k(j)$	Famille des ensembles contenant l'entité j ayant un mouvement τ-dense dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$\overline{\mathcal{W}}_k(j)$	Famille des ensembles contenant l'entité j ayant un mouvement τ-dense maximal dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$D_k(j)$	Ensemble des entités de A_k pouvant être impliqué dans un mouvement τ-dense avec l'entité j dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$J_k(j)$	Ensemble des entités de $D_k(j)$ pour lesquelles j appartient à tous les mouvements τ-dense maximaux dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$L_k(j)$	Ensemble des entités de $D_k(j)$ pour lesquelles j n'appartient pas à tous les mouvements τ-dense maximaux dans l'intervalle de temps $[k-1,k]$ (Section 3.3)
$C_k(j)$	Ensemble I_k, U_k, M_k auquel l'entité j appartient (Section 3.3.5)
$E_k(j)$	Classe d'équivalence de l'entité j(Section 3.3.5).

On cherche à résoudre le problème 2 de manière moins coûteuse. Comme nous l'avons vu précédemment, la définition de ce problème repose sur la décomposition des états du système en ensembles ayant des mouvements r-cohérents. Ainsi la construction de ces ensembles est primordiale pour la résolution du problème 2.

On définit alors les trois familles suivantes $\mathcal{M}_k(j)$, $\mathcal{W}_k(j)$ et $\overline{\mathcal{W}}_k(j)$ représentant respectivement l'ensemble de tous les sous-ensembles de A_k ayant un mouvement r-cohérent impliquant j, l'ensemble de tous les sous-ensembles de A_k ayant un mouvement τ-dense impliquant j et l'ensemble de tous les sous-ensembles de A_k ayant un mouvement τ-dense maximal impliquant j. Formellement, ces trois familles sont définies comme suit :

$$\mathcal{M}_k(j) = \{B \subseteq A_k \mid j \in B \text{ et } B \text{ a un mouvement } r\text{-cohérent}\}$$

$$\mathcal{W}_k(j) = \{B \in \mathcal{M}_k(j) \mid B \text{ a un mouvement } \tau\text{-dense}\}$$

$$\overline{\mathcal{W}}_k(j) = \{B \in \mathcal{M}_k(j) \mid B \text{ a un mouvement } \tau\text{-dense maximal}\}\,.$$

Les théorèmes présentés dans cette section ne s'appuient que sur cette connaissance des mouvements r-cohérents maximaux impliquant chaque entité de A_k. En particulier,

le théorème 2 fournit une condition nécessaire et suffisante pour l'appartenance à I_k. Les théorèmes 3 et 4 décrivent respectivement une condition nécessaire et une condition nécessaire et suffisante pour l'appartenance à M_k. Enfin, le corollaire 3 fournit une condition nécessaire et suffisante pour l'appartenance à U_k. L'ensemble des notations utilisées dans cette section est résumé dans le tableau 3.3.

3.3.1 Condition nécessaire et suffisante pour l'appartenance à I_k

Le théorème 2 suivant illustre qu'en l'absence d'un nombre suffisant d'entités situées dans le voisinage de j ayant une trajectoire similaire à celle de j, l'entité j appartient nécessairement à une anomalie isolée. Dans ce cas, l'entité j a perçu une défaillance due à une faute isolée.

Théorème 2 *Pour tout $k \geq 1$ et pour tout $j \in A_k$, on a l'équivalence suivante :*

$$\overline{\mathcal{W}}_k(j) = \emptyset \Longleftrightarrow j \in I_k.$$

Preuve ($\Rightarrow$) Soit $j \in A_k$. Supposons $\overline{\mathcal{W}}_k(j) = \emptyset$. Pour tout ensemble $B \subseteq [\![1, n]\!]$ ayant un mouvement r-cohérent, il existe un ensemble $B' \subseteq [\![1, n]\!]$ tel que $B \subseteq B'$ et B' a un mouvement r-cohérent maximal (voir la définition 5 et la remarque 1). Par conséquent, on a $\overline{\mathcal{W}}_k(j) = \emptyset \Rightarrow \mathcal{W}_k(j) = \emptyset$. De plus, d'après la définition 6, j appartient à un mouvement non τ-dense et donc d'après la définition 7, pour toute partition d'anomalie $\mathcal{P}_k$, on a $|\mathcal{P}_k(j)| \leq \tau$. En conséquence, d'après la définition 9, j ne peut être impacté que par une anomalie isolée, et donc $j \in I_k$.

($\Leftarrow$) Montrons à présent que $\overline{\mathcal{W}}_k(j) \neq \emptyset \Longrightarrow j \notin I_k$. Supposons $\overline{\mathcal{W}}_k(j) \neq \emptyset$ et $j \in A_k$. Soit $B \in \overline{\mathcal{W}}_k(j)$, on a alors $|B| > \tau$, $j \in B$ et B a un mouvement r-cohérent maximal. D'après le lemme 1, l'algorithme 1 construit une partition d'anomalies. On exécute donc cet algorithme afin de construire une partition d'anomalie dans laquelle j appartient à une anomalie massive. Initialement, on choisit j et un ensemble $B \in \overline{\mathcal{W}}_k(j)$. Par conséquent, B est un élément de la partition finale, on a donc $\mathcal{P}_k(j) = B$. On a ainsi une partition d'anomalies pour laquelle $\mathcal{P}_k(j) > \tau$. D'après la relation (3.2), $j \notin I_k$. □

3.3.2 Condition suffisante pour l'appartenance à M_k

Nous avons montré dans la section précédente, que s'il n'y a pas assez d'entités dans le voisinage de j ayant une trajectoire similaire à celle de j alors j appartient nécessairement à une anomalie isolée.

Nous cherchons à présent à déterminer les conditions d'appartenance à M_k. Nous fournissons dans cette partie une condition suffisante pour l'appartenance à M_k et dans la partie suivante 3.3.3 une condition nécessaire et suffisante pour l'appartenance à M_k.

Supposons à présent que $\overline{\mathcal{W}}_k(j) \neq \emptyset$. Cela signifie qu'il existe un nombre supérieur à τ d'autres entités ayant perçu des variations de mesures similaires. On note $D_k(j)$

l'ensemble de ces entités. Cet ensemble est alors défini par :

$$D_k(j) = \bigcup_{B \in \overline{\mathcal{W}}_k(j)} B.$$

On décompose les éléments de $D_k(j)$ en deux sous-ensembles $J_k(j)$ et $L_k(j)$. Le sous-ensemble $J_k(j)$ contient toutes les entités ℓ de $D_k(j)$ pour lesquelles j appartient à chacun des ensembles ayant un mouvement r-cohérent maximal impliquant ℓ. De manière similaire, le sous-ensemble $L_k(j)$ contient toutes les entités ℓ de $D_k(j)$ pour lesquelles il existe au moins un ensemble contenant ℓ ayant un mouvement r-cohérent maximal auquel j n'appartient pas. Ces deux ensembles sont complémentaires dans $D_k(j)$. On a :

$$J_k(j) = \{\ell \in D_k(j) \mid \forall B \in \overline{\mathcal{W}}_k(\ell),\ j \in B\}, \tag{3.3}$$

$$L_k(j) = \{\ell \in D_k(j) \mid \exists B \in \overline{\mathcal{W}}_k(\ell),\ j \notin B\}. \tag{3.4}$$

Exemple 8 *La figure 3.5 illustre cette décomposition pour l'entité labellisée* 4. *On considère que toutes les entités représentées appartiennent à* A_k *et* $\tau = 2$. *Sur la figure 3.5(a), on a* $S = \{1, 2, 3, 4, 5\}$. *Dans cette configuration, on a* $\overline{\mathcal{W}}_k(4) = \{\{1, 2, 3, 4\}, \{2, 4, 5\}\}$ *et donc* $D_k(4) = \{1, 2, 3, 4, 5\}$. *Par définition, l'entité* 4 *appartient à* $J_k(4)$. *Les entités* $1, 3$ *appartiennent à un unique mouvement* r*-cohérent maximal* τ*-dense* $B_1 = \{1, 2, 3, 4\}$. B_1 *contient l'entité* 4 *et donc on a* $1, 3 \in J_k(4)$. *De la même manière, l'entité* 5 *n'appartient qu'à un unique ensemble ayant un mouvement* r*-cohérent maximal* τ*-dense* $B_2 = \{1, 2, 3, 4, 5\}$. *Celui-ci contient* 4 *et donc* $5 \in J_k(4)$. *Le même raisonnement s'applique à l'entité* 2. *On a donc* $J_k(4) = \{1, 2, 3, 4, 5\}$ *et* $L_k(4) = \emptyset$. *À l'inverse, dans la figure 3.5(b), on a* $S = \{1, 2, 3, 4, 5, 6, 7\}$. *Les entités* 6 *et* 7 *n'appartiennent pas au voisinage de l'entité* 4. *On a donc toujours* $\overline{\mathcal{W}}_k(4) = \{\{1, 2, 3, 4\}, \{2, 4, 5\}\}$. *À présent, l'entité* 5 *appartient à deux ensembles ayant des mouvements* r*-cohérents maximaux* τ*-dense* $B_2 = \{2, 4, 5\}$ *et* $B_3 = \{5, 6, 7\}$. *Or l'entité* 4 *n'appartient pas à* B_3. *En conséquence, on a* $J_k(4) = \{1, 2, 3, 4\}, L_k(4) = \{5\}$.

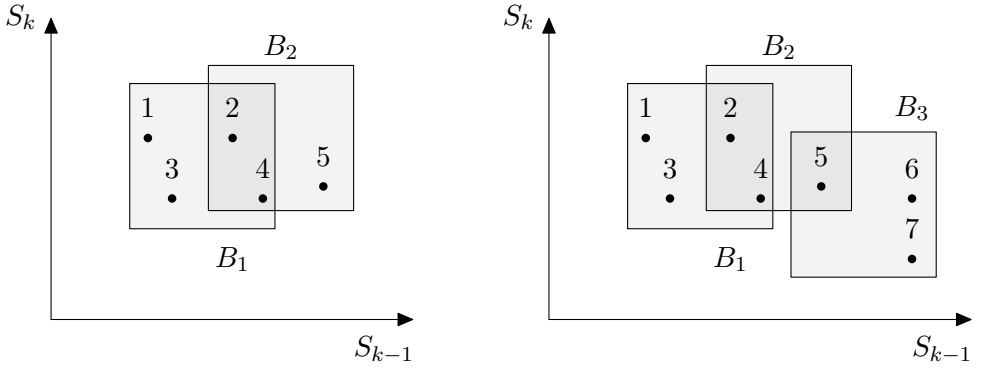
Le théorème suivant s'appuie sur cette décomposition du voisinage afin de fournir une condition suffisante pour l'appartenance d'une entité à M_k.

Théorème 3 *Pour tout* $k \geq 1$ *et tout* $j \in A_k$, *on a l'implication suivante :*

$$\exists B \in \mathcal{W}_k(j) \text{ tel que } B \subseteq J_k(j) \Longrightarrow j \in M_k$$

Preuve Nous prouvons ici par l'absurde ce théorème. Soit $B \subset A_k$ tel que $B \in \mathcal{W}_k(j)$ et $B \subseteq J_k(j)$. Supposons qu'il existe une partition d'anomalies $\mathcal{P}_k = \{C_1, \ldots, C_\ell\}$ telle que $j \in I_{\mathcal{P}_k}$. Deux cas doivent être considérés :

1. Soit $\ell \in B$ tel que $|\mathcal{P}_k(\ell)| > \tau$. D'après la définition 8, $\mathcal{P}_k(\ell)$ a un mouvement τ-dense, et donc $\mathcal{P}_k(\ell) \in \mathcal{W}_k(\ell)$. On note $L \in \overline{\mathcal{W}}_k(\ell)$, tel que $\mathcal{P}_k(\ell) \subseteq L$. D'après la remarque 1, cet élément existe. Par hypothèse du théorème $B \subseteq J_k(j)$ et donc



(a) $D_k(4) = \{1,2,3,4,5\}$, $J_k(4) = \{1,2,3,4,5\}$, et $L_k(4) = \emptyset$, pour $\tau = 2$

(b) $D_k(4) = \{1,2,3,4,5\}$, $J_k(4) = \{1,2,3,4\}$, et $L_k(4) = \{5\}$, pour $\tau = 2$

FIGURE 3.5 – Décomposition du voisinage de l'entité 4 en $J_k(4)$ et $L_k(4)$.

pour tout élément L' de $\overline{\mathcal{W}}_k(\ell)$, j appartient à L'. En particulier, $L \in \overline{\mathcal{W}}_k(\ell)$ on a donc $j \in L$.

Par hypothèse, $j \in I_{\mathcal{P}_k}$ et donc $j \notin \mathcal{P}_k(\ell)$. De plus, on a $\mathcal{P}_k(\ell) \cup \{j\} \subseteq L$ et donc $\mathcal{P}_k(\ell) \cup \{j\}$ a un mouvement r-cohérent. De plus, on a $|\mathcal{P}_k(\ell) \cup \{j\}| > \tau + 1$, donc $\mathcal{P}_k(\ell) \cup \{j\}$ a un mouvement τ-dense, ce qui contredit la condition C2 de la définition 8 de la partition d'anomalies.

2. Supposons à présent que pour tout $\ell \in B$, on ait $|\mathcal{P}_k(\ell)| \leq \tau$. On a alors $B \subseteq \bigcup_{\ell \in B} \mathcal{P}_k(\ell) \subseteq \bigcup_{|C_i| \leq \tau} C_i$. D'autre part, on a $B \in \mathcal{W}_k(j)$, ce qui signifie que B a un mouvement τ-dense, contredisant ainsi la condition C1 de la définition 8 de la partition d'anomalies.

Les deux cas considérés mènent à une contradiction. On a donc $\exists B \in \mathcal{W}_k(j)$ tel que $B \subseteq J_k(j) \Longrightarrow j \in M_k$. □

Corollaire 2 *Pour tout $k \geq 1$ et pour tout $j \in A_k$, on a l'implication suivante :*

$$\exists B \in \overline{\mathcal{W}}_k(j) \text{ tel que } |B \cap J_k(j)| > \tau \Longrightarrow j \in M_k$$

Preuve Soit $B \in \overline{\mathcal{W}}_k(j)$. D'après la remarque 1 tout sous-ensemble de B a aussi un mouvement r-cohérent. C'est le cas en particulier de l'ensemble $B \cap J_k(j) \subseteq B$. D'autre part, si on a $|B \cap J_k(j)| > \tau$ alors $B \cap J_k(j) \in \mathcal{W}_k(j)$. Par application du théorème 3, on a $j \in M_k$. □

Le corollaire 2 découle directement du théorème 3. Néanmoins, son application est plus facile d'un point de vue algorithmique. En effet, pour chaque ensemble $B \in \overline{\mathcal{W}}_k(j)$, tous les sous-ensembles $B' \subseteq B, |B'| > \tau$ appartiennent à $\mathcal{W}_k(j)$. Par conséquent, le cardinal de $\overline{\mathcal{W}}_k(j)$ est inférieur à celui de $\mathcal{W}_k(j)$, et donc cette condition est moins coûteuse à évaluer.

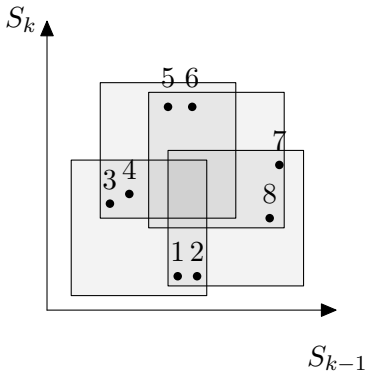


FIGURE 3.6 – Configuration du système où les éléments $j \in [\![1, 8]\!]$ appartiennent à M_k sans qu'un ensemble d'entités de $J_k(j)$ ayant un mouvement dense puisse être exhibé. Ici, le seuil de densité τ vaut 3.

3.3.3 Condition nécessaire et suffisante pour l'appartenance à M_k

Comme nous venons de le voir, le théorème 3 propose une condition nécessaire pour l'appartenance d'une entité à M_k. Comme il ne s'agit pas d'une équivalence, certaines entités appartenant à $M_k = \{\ell \in A_k \mid \forall \mathcal{P}_k, |\mathcal{P}_k(\ell)| > \tau\}$ ne vérifient pas ce théorème.

Exemple 9 *Considérons la situation illustrée par la figure 3.6. Le système est composé de 8 entités* $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ *percevant une défaillance et supposons* $\tau = 3$. *Considérons l'entité* 1, *on a* $\overline{\mathcal{W}}_k(1) = \{\{1, 2, 3, 4\}, \{1, 2, 7, 8\}\}$ *et donc* $D_k(1) = \{1, 2, 3, 4, 7, 8\}$. *Un raisonnement analogue s'applique aux autres entités du fait de la symétrie de cette configuration. Par définition,* $J_k(1) = \{1, 2\}$ *et* $L_k(1) = \{3, 4, 7, 8\}$. *Comme* $|J_k(1)| < \tau$, *il n'existe pas d'ensemble formé d'éléments de* $J_k(1)$ *ayant un mouvement* τ*-dense. Par conséquent, le théorème 3 ne s'applique pas.*

Néanmoins, l'entité 1 *appartient à* M_k *: quelle que soit la partition d'anomalies* $\mathcal{P}_k$ *considérée, on a toujours* $|\mathcal{P}_k(1)| > \tau$. *En effet, il n'existe que deux partitions d'anomalies pour cette configuration :*

— $\mathcal{P}_k = \{\{1, 2, 7, 8\}, \{3, 4, 5, 6\}\}$

— $\mathcal{P}'_k = \{\{1, 2, 3, 4\}, \{5, 6, 7, 8\}\}$.

On a alors $|\mathcal{P}_k(1)| = |\mathcal{P}'_k(1)| > \tau$ *et donc* $1 \in M_k$ *alors qu'il est impossible d'exhiber un ensemble d'entités de* $J_k(1)$ *ayant un mouvement* τ*-dense.*

Le théorème suivant fournit une condition nécessaire et suffisante pour l'appartenance à M_k.

Théorème 4 *Pour tout $k \geq 1$ et tout $j \in A_k$, on a $j \in M_k$ si et seulement si pour toute famille non vide $\mathcal{F}$ constituée d'ensembles deux à deux disjoints telle que $\mathcal{F} \subseteq \{B \in \bigcup_{\ell \in L_k(j)} \mathcal{W}_k(\ell) \mid j \notin B\}$ on a :*

$$\left(\exists B \in \mathcal{W}_k(j) \text{ tel que } B \subseteq D_k(j) \setminus \bigcup_{C \in \mathcal{F}} C\right) \text{ ou } \left(\exists C \in \mathcal{F} \text{ tel que } C \cup \{j\} \in \mathcal{W}_k(j)\right).$$

Preuve Nous allons prouver ce théorème par contraposée.

($\Rightarrow$)Supposons $j \notin M_k$. Cela signifie qu'il existe une partition d'anomalie $\mathcal{P}_k = \{C_1, \ldots, C_\ell\}$ telle que $|\mathcal{P}_k(j)| \leq \tau$. Considérons la collection suivante : $\mathcal{F}_1 = \{C \in \mathcal{P}_k \mid |C| > \tau\}$.

Par définition de la partition d'anomalie, tous les éléments de $\mathcal{F}_1$ sont deux à deux disjoints. D'autre part, d'après la condition C2 de la définition 8 de la partition d'anomalie, on a pour tout i dans $[\![1, \ell]\!]$, si C_i a un mouvement τ-dense alors pour tout $B \subseteq \bigcup_{|C_j| \leq \tau} C_j, B \cup C_i$ n'a pas un mouvement r-cohérent. En particulier, cela signifie que pour tout i dans $[\![1, \ell]\!]$, si $|C_i| > \tau$ alors $C_i \cup \{j\}$ n'a pas un mouvement r-cohérent. L'ensemble $C_i \cup \{j\}$ n'a donc pas un mouvement r-cohérent τ-dense. On a $C_i \cup \{j\} \notin \mathcal{W}_k(j)$. Par conséquent, j n'appartient à aucun élément de $\mathcal{F}_1$.

De plus, d'après la construction de $\mathcal{F}_1$, toutes les parties de $\mathcal{P}_k$ ayant un mouvement τ-dense appartiennent à $\mathcal{F}_1$. Par conséquent, pour tout $\ell \in D_k(j) \setminus \bigcup_{C \in \mathcal{F}_1} C$, on a $|\mathcal{P}_k(\ell)| \leq \tau$. Cela signifie que $D_k(j) \setminus \bigcup_{C \in \mathcal{F}_1} C \subseteq \bigcup_{|C_i| \leq \tau} C_i$. D'après la condition C1 de la définition 8 de la partition d'anomalie, tout sous-ensemble B de $\bigcup_{|C_i| \leq \tau} C_i$ n'a pas de mouvement τ-dense. Par conséquent, on a $B \notin \mathcal{W}_k(j)$. Cela signifie que l'on a $\forall B \in \mathcal{W}_k(j), B \not\subseteq D_k(j) \setminus \bigcup_{C \in \mathcal{F}_1} C$, ce qui conclut cette partie de la preuve.

($\Leftarrow$) Nous devons considérer deux cas pour prouver la réciproque : $\mathcal{W}_k(j) = \emptyset$ et $\mathcal{W}_k(j) \neq \emptyset$.

Cas $\mathcal{W}_k(j) = \emptyset$: le théorème 2 s'applique et donc on a $j \in I_k$. Par conséquent on a $j \notin M_k$.

Cas $\mathcal{W}_k(j) \neq \emptyset$: supposons qu'il existe une famille d'ensembles deux à deux disjoints $\mathcal{F}_2 \subseteq \{B \in \bigcup_{\ell \in L_k(j)} \mathcal{W}_k(\ell) \mid j \notin B\}$ vérifiant les deux propositions suivantes :

$$\forall B \in \mathcal{W}_k(j), B \not\subseteq D_k(j) \setminus \bigcup_{C \in \mathcal{F}_2} C, \tag{3.5}$$

$$\forall C \in \mathcal{F}_2, C \cup \{j\} \notin \mathcal{W}_k(j). \tag{3.6}$$

Nous montrons que sous ces hypothèses, on peut construire une partition d'anomalie $\mathcal{P}_k$ telle que $|\mathcal{P}_k(j)| \leq \tau$.

Soit $\mathcal{P}_k$ une partition d'anomalie telle que tout élément de $\mathcal{F}_2$ soit une partie de $\mathcal{P}_k$. Cela signifie que pour tout ensemble C de $\mathcal{F}_2$, C appartient à $\mathcal{P}_k$. D'après la relation (3.5), tous les ensembles impliquant j ayant un mouvement τ-dense contiennent des entités appartenant à un élément de $\mathcal{F}_2$. De plus, d'après la relation (3.6), j ne peut être impliqué dans un mouvement τ-dense avec un élément de $\mathcal{F}_2$. Ainsi, par construction de $\mathcal{P}_k$, on a $|\mathcal{P}_k(j)| \leq \tau$. Par conséquent, $j \notin M_k$, ce qui conclut la seconde partie

de la preuve. □

Exemple 10 *Revenons à présent à l'exemple décrit dans la figure 3.6. On a* $\tau = 3$ *et on considère l'entité* 1*. On a* $D_k(1) = \{1, 2, 3, 4, 7, 8\}$ *et* $L_k(1) = \{3, 4, 7, 8\}$*. On a alors*

$$\{B \in \mathcal{W}_k(\ell) \mid \ell \in L_k(1), 1 \notin B\} = \{\{3, 4, 5, 6\}, \{5, 6, 7, 8\}\}.$$

Deux cas sont alors à considérer :

1. $\mathcal{F}_1 = \{\{3, 4, 5, 6\}\}$*. Dans ce cas,* $B_1 = D_k(1) \setminus \left(\bigcup_{C \in \mathcal{F}_1} C\right) = \{1, 2, 8, 7\}$*. Le seul sous-ensemble de* B_1 *ayant un mouvement* τ*-dense est* B_1*. L'entité* 1 *appartient à* B_1 *et donc appartient à une anomalie massive dans cette configuration.*
2. $\mathcal{F}_2 = \{\{5, 6, 7, 8\}\}$*. Dans ce cas,* $B_2 = D_k(1) \setminus \left(\bigcup_{C \in \mathcal{F}_2} C\right) = \{1, 2, 3, 4\}$*. Le seul sous-ensemble de* B_2 *ayant mouvement* τ*-dense est* B_2*. L'entité* 1 *appartient à* B_2 *et donc appartient à une anomalie massive dans cette configuration.*

De plus, il n'existe pas d'autre famille d'ensembles deux à deux disjoints incluse dans $\{B \in \mathcal{W}_k(\ell) \mid \ell \in L_k(1), 1 \notin B\}$*. Toutes les collections ont été testées, on a donc* $1 \in M_k$*.*

3.3.4 Condition nécessaire et suffisante pour l'appartenance à U_k

Nous concluons à présent cette partie en fournissant une condition nécessaire et suffisante d'appartenance à U_k. Ce corollaire découle directement des théorèmes 3 et 4.

Corollaire 3 *Pour tout instant* $k \geq 1$ *et pour tout* $j \in A_k$*, on a* $j \in U_k$ *si et seulement si* $\overline{\mathcal{W}}_k(j) \neq \emptyset$ *et s'il existe une famille d'ensembles deux à deux disjoints* $\mathcal{F} \subseteq \{B \in \mathcal{W}_k(\ell) \mid \ell \in L_k(j), j \notin B\}$ *satisfaisant la proposition suivante :*

$$\left(\forall B \in \mathcal{W}_k(j) :\ B \not\subseteq D_k(j) \setminus \bigcup_{C \in \mathcal{F}} C\right) \wedge \left(\forall C \in \mathcal{F} :\ C \cup \{j\} \notin \mathcal{W}_k(j)\right).$$

Preuve La preuve de ce corollaire est une application directe du second cas de la preuve du théorème 4. □

3.3.5 Équivalence de caractérisations

Nous venons d'exhiber des conditions permettant à toute entité percevant une défaillance dans le système de déterminer si la défaillance perçue est due à une faute isolée ou une faute massive. De plus, nous avons montré que l'occurrence de diverses fautes concomitantes dans le système peut amener le système dans une configuration où certaines entités sont incapables de déterminer avec certitude le type de faute ayant engendré la défaillance perçue.

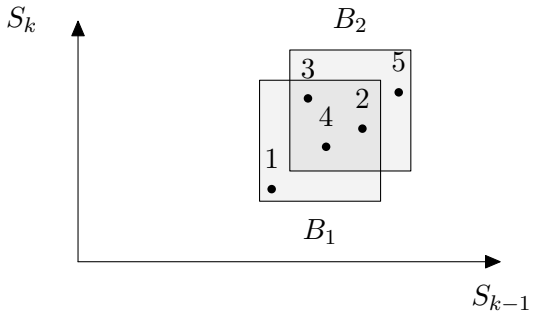


FIGURE 3.7 – Calculs équivalents

Dans cette section, nous proposons une nouvelle caractérisation visant à réduire le nombre de calculs nécessaires pour déterminer pour chaque entité ayant perçu une défaillance, le type de faute l'ayant provoquée.

Nous illustrons l'intuition que nous allons développer au cours de cette section par l'exemple suivant.

Exemple 11 *Considérons à présent la situation illustrée dans la figure 3.7. Le système composé de 5 entités* $\{1,2,3,4,5\}$. *Supposons que toutes ces entités percoivent une défaillance dans l'intervalle de temps* $[k-1,k]$ *et* $\tau = 3$. *Les deux ensembles* $B_1 = \{1,2,3,4\}$ *et* $B_2 = \{2,3,4,5\}$ *ont un mouvement* τ*-dense et sont maximaux.*

Cas de l'entité 1 *: L'entité* 1 *appartient à un unique ensemble ayant un mouvement* τ*-dense maximal :* B_1. *Les entités* $2,3,4$ *appartiennent aussi à l'ensemble* B_2. *Par conséquent, on a* $J_k(1) = \{1\}$ *et* $L_k(1) = \{2,3,4\}$. *Par application du corollaire 3, l'entité* 1 *appartient à* U_k.

Cas de l'entité 2 *: L'entité* 2 *appartient aux deux ensembles* B_1 *et* B_2. *On a alors* $J_k(2) = \{1,2,3,4,5\}$. *Par application du théorème 3, l'entité* 2 *appartient à* M_k.

Cas de l'entité 3 *: L'entité* 3 *appartient aux deux ensembles* B_1 *et* B_2. *On a alors* $J_k(3) = \{1,2,3,4,5\}$. *Par application du théorème 3, l'entité* 3 *appartient à* M_k.

Cas de l'entité 4 *: L'entité* 4 *appartient aux deux ensembles* B_1 *et* B_2. *On a alors* $J_k(4) = \{1,2,3,4,5\}$. *Par application du théorème 3, l'entité* 4 *appartient à* M_k.

Cas de l'entité 5 *: L'entité* 5 *appartient à un unique ensemble ayant un mouvement* τ*-dense maximal :* B_2. *Les entités* $2,3,4$ *appartiennent aussi à l'ensemble* B_1. *Par conséquent, on a* $J_k(5) = \{5\}$ *et* $L_k(5) = \{2,3,4\}$. *Par application du corollaire 3, l'entité* 5 *appartient à* U_k.

On a :

$$I_k = \emptyset, M_k = \{2,3,4\} \text{ et } U_k = \{1,5\}.$$

Dans cet exemple, on a $J_k(2) = J_k(3) = J_k(4)$. On applique alors pour chacune de ces entités le théorème 3 afin de déterminer le type de faute ayant provoqué la défaillance perçue. On a $M_k = \{2,3,4\}$. On cherche à présent à mutualiser les calculs nécessaires à la caractérisation afin de n'appliquer celle-ci qu'une seule fois pour l'ensemble $\{2,3,4\}$.

Théorème 5 *Pour tout $k \geq 1$ et pour tout $j, \ell \in A_k$, on a l'équivalence suivante :*

$$\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell) \Longleftrightarrow J_k(j) = J_k(\ell).$$

Preuve ($\Longrightarrow$) Soient $j, \ell \in A_k$ tels que $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$. On a alors $\forall B \in \overline{\mathcal{W}}_k(j), B \in \overline{\mathcal{W}}_k(\ell)$. De plus, $\forall B \in \overline{\mathcal{W}}_k(j), \ell \in B$, et donc $\ell \in J_k(j)$. De la même manière, on a $j \in J_k(\ell)$. On a donc $J_k(j) = J_k(\ell)$.

($\Longleftarrow$) Soient $j, \ell \in A_k$ tels que $J_k(j) = J_k(\ell)$. Par définition de l'ensemble $J_k(j)$, on a $\forall B \in \overline{\mathcal{W}}_k(\ell), j \in B$. De plus, B a un mouvement maximal, donc $B \in \overline{\mathcal{W}}_k(j)$. On a donc $\overline{\mathcal{W}}_k(\ell) \subseteq \overline{\mathcal{W}}_k(j)$. De la même manière, on a $\forall B \in \overline{\mathcal{W}}_k(j), \ell \in B$. De plus, B a un mouvement maximal, donc $B \in \overline{\mathcal{W}}_k(\ell)$. On a donc $\overline{\mathcal{W}}_k(j) \subseteq \overline{\mathcal{W}}_k(\ell)$ et donc $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$. □

Corollaire 4 *Pour tout $k \geq 1$ et pour tout $j, \ell \in A_k$, on a l'équivalence suivante :*

$$\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell) \Longleftrightarrow L_k(j) = L_k(\ell).$$

Preuve Soient $j, \ell \in A_k$ tels que $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$. Par définition, on a $D_k(j) = \cup_{B \in \overline{\mathcal{W}}_k(j)} B$. On a donc $D_k(j) = D_k(\ell)$. D'autre part, l'ensemble $L_k(j)$ est défini comme le complémentaire de $J_k(j)$ dans $D_k(j)$. D'après le théorème 5, on a $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell) \Longleftrightarrow J_k(j) = J_k(\ell)$. Par conséquent, on a donc aussi $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell) \Longleftrightarrow L_k(j) = L_k(\ell)$. □

On définit la relation suivante.

Définition 11 *Pour tout $k \geq 1$ et pour tout $j, \ell \in A_k$, on définit la relation*

$$j \equiv \ell \Longleftrightarrow \overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell).$$

Propriété 2 *La relation $\equiv$ est une relation d'équivalence dans A_k. On note $E_k(j)$ la classe d'équivalence de j dans A_k. On a $E_k(j) = \{\ell \in A_k \mid \ell \equiv j\}$.*

Preuve La relation $\equiv$ est une relation binaire et celle-ci est réflexive : par définition, pour tout $j \in A_k$, on a $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(j)$ et donc $j \equiv j$.

D'autre part, la relation $\equiv$ est symétrique : pour tous $j, \ell \in A_k$ tels que $j \equiv \ell$, on a $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$. Par conséquent, on a $\overline{\mathcal{W}}_k(\ell) = \overline{\mathcal{W}}_k(j)$ et donc $\ell \equiv j$.

Enfin, la relation $\equiv$ est transitive : pour tous $j, \ell, p \in A_k$ tels que $j \equiv \ell$ et $\ell \equiv p$, on a $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$ et $\overline{\mathcal{W}}_k(\ell) = \overline{\mathcal{W}}_k(p)$. Par conséquent, on a donc $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(p)$ et donc $j \equiv p$.

La relation $\equiv$ est donc une relation binaire réflexive, symétrique et transitive, il s'agit d'une relation d'équivalence. □

On partitionne ici l'ensemble A_k en trois sous-ensembles disjoints I_k, M_k et U_k. Pour tout $j \in A_k$, on note $C_k(j)$ le sous-ensemble auquel j appartient. On a alors :

Théorème 6 *Pour tout $k \geq 1$ et pour tout $j, \ell \in A_k$, on a*

$$j \equiv \ell \Longrightarrow C_k(j) = C_k(\ell).$$

Preuve Soient $j, \ell \in A_k$ tels que $j \equiv \ell$, on a alors $\overline{\mathcal{W}}_k(j) = \overline{\mathcal{W}}_k(\ell)$. Deux cas sont à considérer : $\overline{\mathcal{W}}_k(j) = \emptyset$ et $\overline{\mathcal{W}}_k(j) \neq \emptyset$.

Cas $\overline{\mathcal{W}}_k(j) = \emptyset$: On a $\overline{\mathcal{W}}_k(\ell) = \emptyset$. Par application du théorème 2, on a $j \in I_k$ et $\ell \in I_k$. On a donc $C_k(j) = C_k(\ell) = I_k$.

Cas $\overline{\mathcal{W}}_k(j) \neq \emptyset$: On a $D_k(j) \neq \emptyset$, $J_k(j) \neq \emptyset$ et $L_k(j) \neq \emptyset$. D'après le théorème 5 et le corollaire 4, on a $J_k(j) = J_k(\ell)$ et $L_k(j) = L_k(\ell)$. Par conséquent, si $j \in M_k$, le théorème 4 s'applique de la même manière pour ℓ et on a alors $C_k(j) = C_k(\ell) = M_k$. De manière analogue, si $j \in U_k$, le corollaire 3 s'applique de la même manière pour ℓ et on a alors $C_k(j) = C_k(\ell) = U_k$. □

L'ensemble des classes d'équivalence des éléments de A_k forme une partition de A_k. D'autre part, d'après le théorème 6, tous les éléments de $E_k(j)$ calculent le même ensemble $C_k(j)$. Il est donc suffisant de caractériser le type de faute pour un représentant j de chaque classe d'équivalence $E_k(j)$ afin de déterminer pour l'ensemble des entités de A_k le type de faute ayant provoqué la défaillance perçue par ces entités permettant ainsi de réduire le coût de calcul total de cette caractérisation. Le corollaire suivant nous permet de calculer l'ensemble $E_k(j)$.

Corollaire 5 *Pour tout $k \geq 1$ et pour tout $j \in A_k$, on a*

$$E_k(j) = \{\ell \in J_k(j) \mid j \in J_k(\ell)\}.$$

Preuve Ce corollaire découle directement du théorème 5 et de la définition 11 de la relation d'équivalence. □

Exemple 12 *Revenons sur la situation illustrée dans la figure 3.7. Le système composé de 5 entités $\{1, 2, 3, 4, 5\}$. Supposons que toutes ces entités percoivent une défaillance dans l'intervalle de temps $[k-1, k]$ et $\tau = 3$. Les deux ensembles $B_1 = \{1, 2, 3, 4\}$ et $B_2 = \{2, 3, 4, 5\}$ ont un mouvement τ-dense et sont maximaux. On a alors :*

- $\overline{\mathcal{W}}_k(1) = \{B_1\}$
- $\overline{\mathcal{W}}_k(2) = \{B_1, B_2\}$
- $\overline{\mathcal{W}}_k(3) = \{B_1, B_2\}$
- $\overline{\mathcal{W}}_k(4) = \{B_1, B_2\}$
- $\overline{\mathcal{W}}_k(5) = \{B_2\}$

Par conséquent, on a $2 \equiv 3 \equiv 4$. On a :

$$E_k(1) = \{1\}, E_k(2) = \{2, 3, 4\} \text{ et } E_k(5) = \{5\}.$$

On effectue la caractérisation pour un représentant de chacune de ces classes. On a alors :

Cas de l'entité 1 : L'entité 1 appartient à un unique ensemble ayant un mouvement τ-dense maximal : B_1. Les entités $2, 3, 4$ appartiennent aussi à l'ensemble B_2. Par conséquent, on a $J_k(1) = \{1\}$ et $L_k(1) = \{2, 3, 4\}$. Par application du corollaire 3, l'entité 1 appartient à U_k.

Cas de l'entité 2 *: L'entité* 2 *appartient aux deux ensembles* B_1 *et* B_2*. On a alors* $J_k(2) = \{1, 2, 3, 4, 5\}$*. Par application du théorème 3, l'entité* 2 *appartient à* M_k*.*

Cas de l'entité 5 *: L'entité* 5 *appartient à un unique ensemble ayant un mouvement* τ*-dense maximal :* B_2*. Les entités* $2, 3, 4$ *appartiennent aussi à l'ensemble* B_1*. Par conséquent, on a* $J_k(5) = \{5\}$ *et* $L_k(5) = \{2, 3, 4\}$*. Par application du corollaire 3, l'entité* 5 *appartient à* U_k*.*

Les entités 3 *et* 4 *appartiennent à la classe d'équivalence de l'entité* 2*. Par application du théorème 6, les entités* 3 *et* 4 *appartiennent alors à* M_k*. On a :*

$$I_k = \emptyset, M_k = \{2, 3, 4\} \text{ et } U_k = \{1, 5\}.$$

Nous avons présenté dans cette section des conditions permettant à toute entité percevant une défaillance dans le système de déterminer avec certitude si la défaillance perçue est due à une faute isolée ou une faute massive ou s'il s'agit d'un état indéterminé. Ces conditions nous permettent alors de résoudre le problème 2.

D'autre part, ces conditions ne s'appuient que sur la connaissance des variations de mesures perçues par d'autres entités proches dans l'espace des qualités. Ainsi, une connaissance locale est suffisante pour déterminer avec certitude le type de faute ayant engendré la défaillance perçue. Cela signifie qu'une connaissance plus large, telle que celle qu'aurait un observateur global du système n'apporte aucune information supplémentaire ni une précision accrue sur le type de faute ayant engendré la défaillance perçue.

Enfin, nous avons exhibé des conditions permettant de définir des entités comme équivalentes, nous permettant ainsi de réduire le nombre d'entités pour lesquelles l'évaluation de ces conditions est nécessaire.

3.4 Conclusion

Nous avons montré dans ce chapitre qu'il est possible de déterminer le type d'erreur impactant le système en s'appuyant uniquement sur des mesures de performances de services effectuées par des entités supervisées consommant ces services. De plus, la caractérisation des erreurs suivant leur impact est faite sans connaissance supplémentaire de la topologie du réseau reliant les différentes entités supervisées au serveur délivrant les différents services.

D'autre part, on a montré dans ce chapitre, qu'une connaissance locale du voisinage dans l'espace des qualités est suffisante pour permettre cette caractérisation. Une connaissance plus large, telle que celle qu'aurait un observateur global du système ne permet pas une meilleure caractérisation.

Ce modèle s'appuie uniquement sur la connaissance du voisinage dans l'espace des qualités des entités percevant des défaillances. Par conséquent, le caractère local des informations requises pour la caractérisation rend cette approche décentralisable par nature. Le chapitre suivant décrit les algorithmes nécessaires à la mise en œuvre pratique des conditions nécessaires à la résolution du problème considéré. De plus, cette approche sera validée et sa pertinence évaluée au travers de multiples simulations.

Bibliographie

[ABL+14a] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola : Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, juin 2014.

[ABL+14b] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand, B. Sericola et G. Straub : Anomaly Characterization Problems. Dans *16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, 2014.

[Hol04] C. C. Holt : Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[Pag54] E. S. Page : Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, juin 1954.

[Win60] P. R. Winters : Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 6:324–342, 1960.

Chapitre 4

Mise en œuvre algorithmique et évaluation

Nous avons montré dans le chapitre précédent qu'il est possible de déterminer le type de faute impactant le système en s'appuyant uniquement sur des mesures de performances de services effectuées par des entités supervisées consommant ces services. Cette caractérisation est effectuée sans connaissance supplémentaire de la topologie du réseau reliant les différentes entités supervisées au serveur délivrant les différents services. Ce modèle s'appuie uniquement sur la connaissance du voisinage dans l'espace des qualités des entités percevant des défaillances. Par conséquent, le caractère local des informations requises pour la caractérisation rend cette approche à la fois parallélisable et garantit la propriété de passage à l'échelle.

Ce chapitre présente tout d'abord les algorithmes nécessaires à l'implémentation des conditions de caractérisation décrites dans le chapitre précédent puis une évaluation de ce modèle.

4.1 Algorithmes

Cette partie est dédiée à la présentation d'algorithmes permettant la mise en œuvre des conditions d'appartenance à I_k, M_k et U_k décrites dans le chapitre précédent. Dans la suite, nous supposons que l'entité supervisée j considérée est telle que $a_k(j) = \texttt{VRAI}$, c'est-à-dire que j appartient à A_k.

4.1.1 Calcul des ensembles ayant un mouvement r-cohérent maximal

Chacune des conditions d'appartenance à I_k, M_k et U_k s'appuie sur la connaissance des mouvements r-cohérents τ-denses impliquant une entité supervisée j. Il est donc nécessaire de construire l'ensemble $\mathcal{W}_k(j)$. Comme nous l'avons vu, cet ensemble peut aisément être calculé à partir de l'ensemble $\overline{\mathcal{W}}_k(j)$.

L'algorithme 2 présente le pseudo-code exécuté par l'entité supervisée j afin de construire la famille $\mathcal{M}_k(j)$ des ensembles contenant j ayant un mouvement r-cohérent

maximal. En particulier, cet ensemble contient tous les ensembles contenant j ayant un mouvement r-cohérent τ-dense maximal. On a $\overline{\mathcal{W}}_k(j) \subseteq \mathcal{M}_k(j)$. Cet algorithme effectue une recherche exhaustive parmi l'ensemble des entités supervisées du système dont la distance qui les sépare de j dans E aux instants $k-1$ et k n'excède pas $2r$. Soit $N_k(j)$ cet ensemble, on a alors :

$$N_k(j) = \{\ell \in A_k \mid \|p_{k-1}(\ell) - p_{k-1}(j)\| \leq 2r \text{ et } \|p_k(\ell) - p_k(j)\| \leq 2r\}.$$

L'algorithme construit pour $i = 1, \ldots, d$ l'ensemble $N_k^{(i)}(j) = \{\ell_0, \ldots, \ell_m\}$ contenant les entités de $N_k(j)$ ordonnées par ordre croissant de leur i-ème coordonnée. Il sélectionne ensuite le sous-ensemble $N_k^{(i,0)}(j) \subseteq N_k^{(i)}(j)$ contenant les entités dont la distance à l'instant k les séparant de l'entité ℓ_0 de $N_k^{(i)}(j)$ n'excède pas $2r$. L'algorithme est exécuté récursivement sur cet ensemble en considérant à présent la $(i+1)$-ème coordonnée.

Lorsque toutes les dimensions ont été parcourues ($i = d$) à l'instant k, l'algorithme recommence cette procédure sur les positions des entités à l'instant $k-1$ et créé ainsi l'ensemble $N_{k-1}^{(i,0)}$. Au dernier niveau de récursivité, l'ensemble $N_{k-1}^{(d,0)}$ contient uniquement les entités vérifiant la relation suivante :

$$\forall \ell_1, \ell_2 \in N_{k-1}^{(d,0)}, \|p_{k-1}(\ell_1) - p_{k-1}(\ell_2)\| \leq 2r \text{ et } \|p_k(\ell_1) - p_k(\ell_2)\| \leq 2r.$$

D'après la définition 2, l'ensemble $N_{k-1}^{(d,0)}$ est r-cohérent. Si $N_{k-1}^{(d,0)}$ n'est inclus dans aucun ensemble de la famille $\mathcal{M}_k(j)$ et n'inclut aucun ensemble de $\mathcal{M}_k(j)$ alors celui-ci est ajouté à $\mathcal{M}_k(j)$. Dans le cas contraire, tous les sous-ensembles de $N_{k-1}^{(d,0)}$ appartenant à $\mathcal{M}_k(j)$ sont supprimés de $\mathcal{M}_k(j)$ avant d'y ajouter $N_{k-1}^{(d,0)}$.

L'algorithme supprime ensuite le premier élément de $N_{k-1}^{(d,0)}$ et sélectionne ensuite le sous-ensemble $N_{k-1}^{(d,1)}$ contenant les entités dont la distance à l'instant k les séparant l'entité ℓ_1 de $N_{k-1}^{(d,1)}$ n'excède pas $2r$ et calcule un nouvel ensemble r-cohérent. Ce processus est répété pour chaque dimension aux deux instants jusqu'à ce que l'entité j soit supprimée de l'ensemble courant. Lorsque toutes les dimensions ont été traitées aux deux instants, l'ensemble $\mathcal{M}_k(j)$ contient tous les mouvements r-cohérents maximaux impliquant l'entité j.

Exemple 13 *Les figures 4.2, 4.3 et 4.4 illustrent le fonctionnement de l'algorithme 2. La figure 4.1 illustre les différents ensembles testés. Le système est composé de 8 entités consommant un unique service (*i.e., *$d = 1$). Considérons l'entité $j = 3$. Cette entité construit tous les ensembles ayant un mouvement r-cohérent maximal la contenant. Les entités 1, 2, 4, 5, 6, 7, 8 sont situées dans le voisinage de l'entité 3 dans l'espace des qualités aux instants $k-1$ et k. On a $N_k(3) = \{1, 2, 3, 4, 5, 6, 7, 8\}$ (voir la figure 4.2(a)).*

Initialement, l'algorithme 2 est exécuté avec :

$$N(3) = N_k(3), i = 0, \ell = 0 \text{ et } \mathcal{M}_k(3) = \emptyset.$$

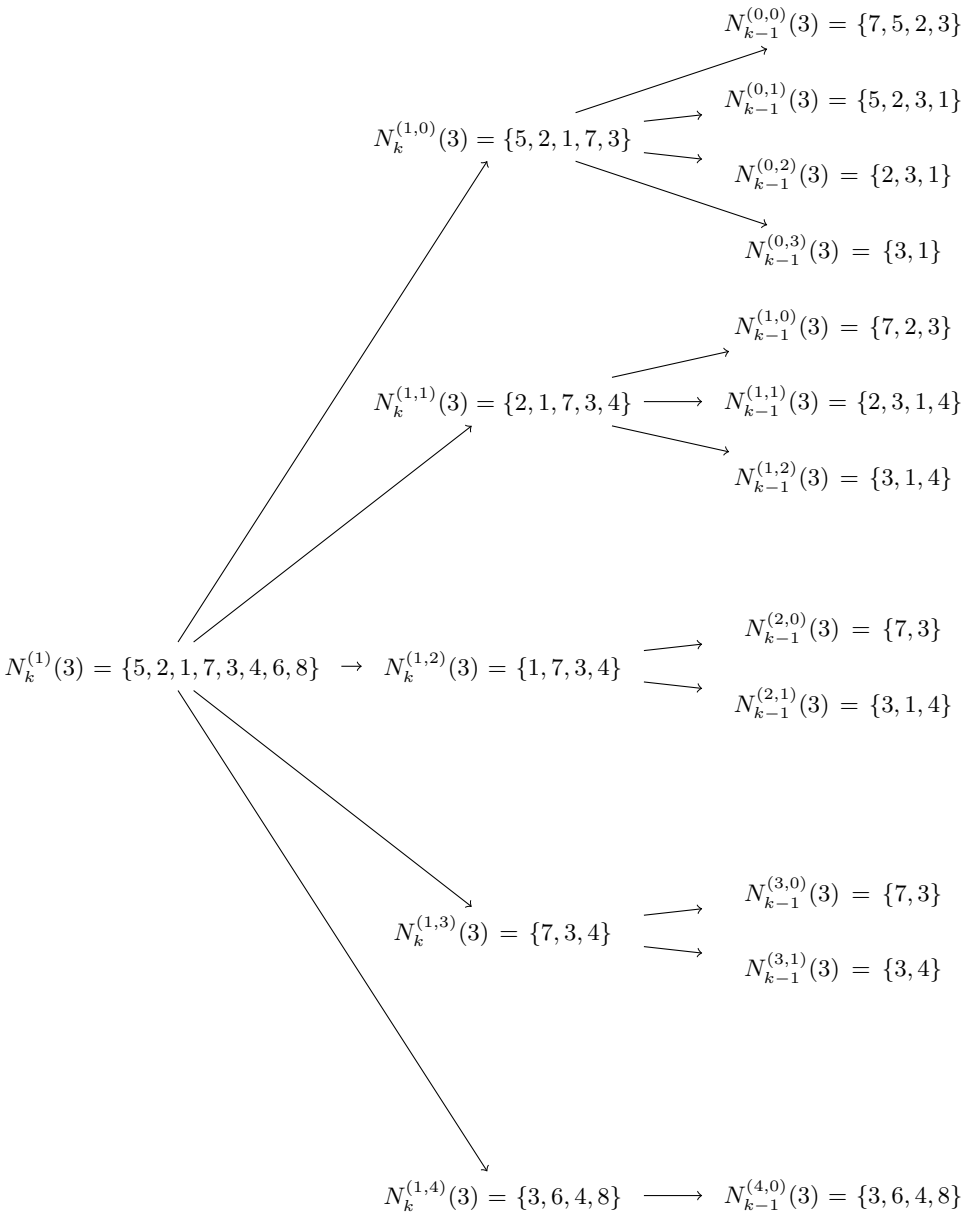
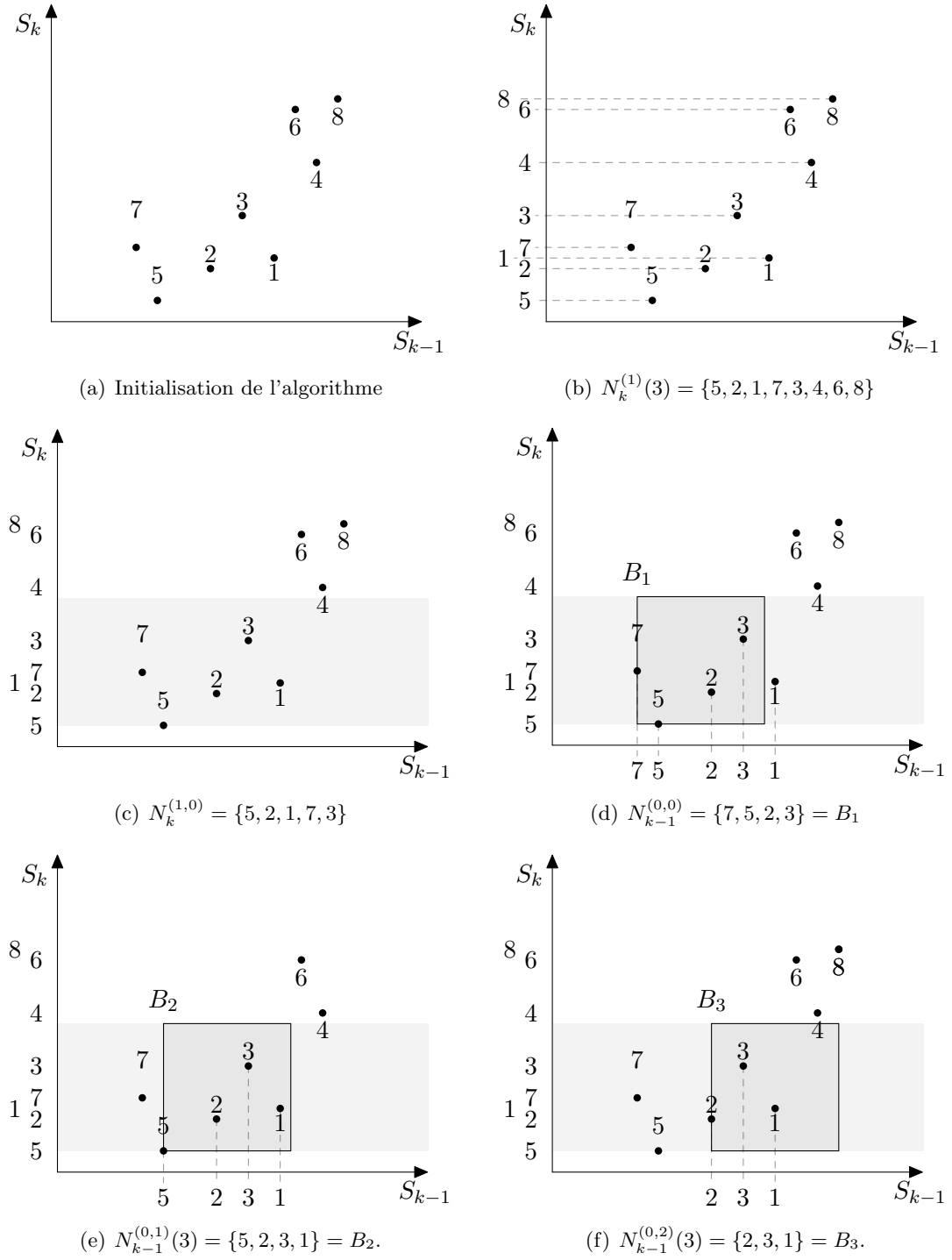


FIGURE 4.1 – Les différents ensembles testés par l'algorithme 2.



(a) Initialisation de l'algorithme

(b) $N_k^{(1)}(3) = \{5, 2, 1, 7, 3, 4, 6, 8\}$

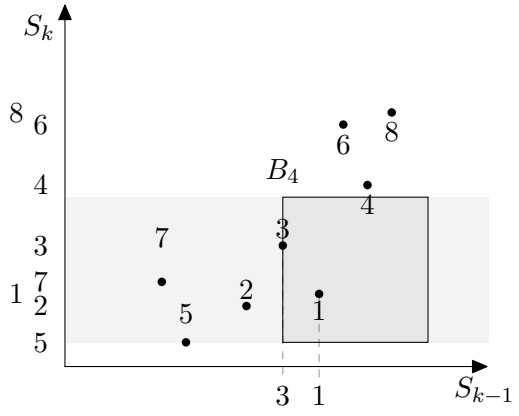
(c) $N_k^{(1,0)} = \{5, 2, 1, 7, 3\}$

(d) $N_{k-1}^{(0,0)} = \{7, 5, 2, 3\} = B_1$

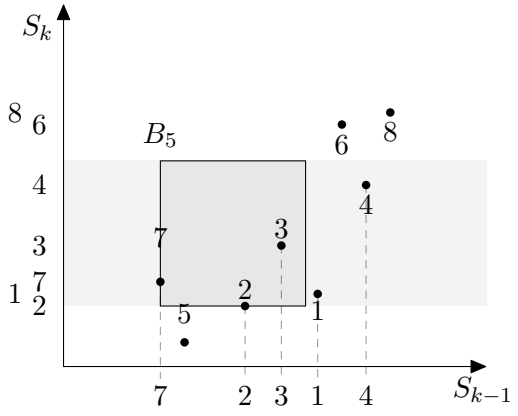
(e) $N_{k-1}^{(0,1)}(3) = \{5, 2, 3, 1\} = B_2$.

(f) $N_{k-1}^{(0,2)}(3) = \{2, 3, 1\} = B_3$.

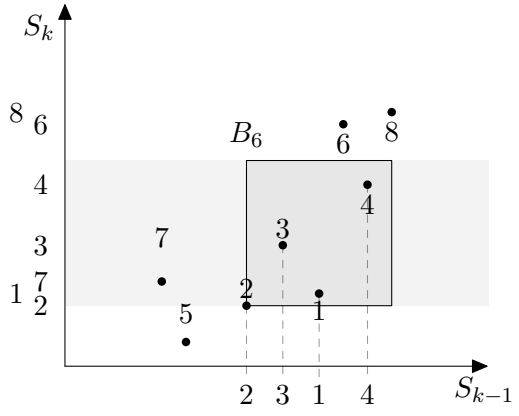
FIGURE 4.2 – Exécution de l'algorithme 2 sur un ensemble de 8 entités.



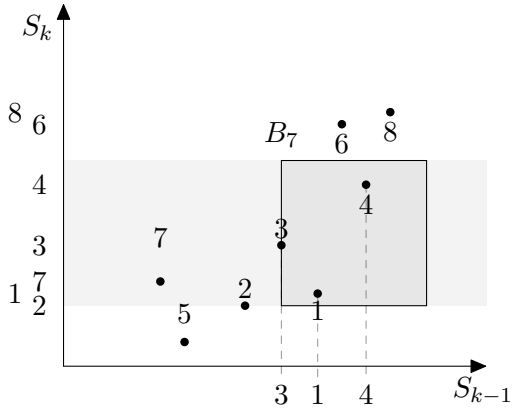
(a) $N_{k-1}^{(0,3)}(3) = \{3, 1\} = B_4$.



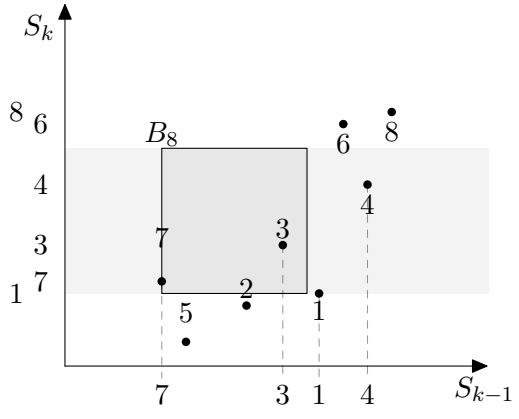
(b) $N_k^{(1,1)}(3) = \{2, 1, 7, 3, 4\}$, $N_{k-1}^{(1,0)}(3) = \{7, 2, 3\} = B_5$



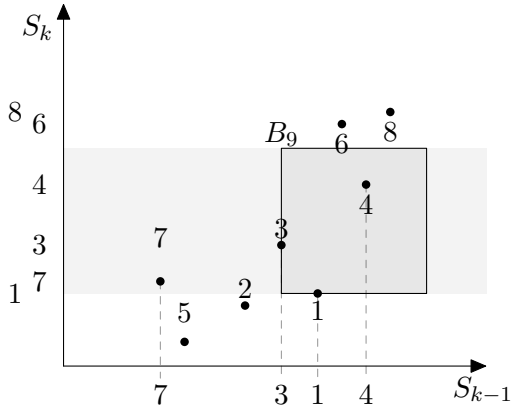
(c) $N_{k-1}^{(1,1)}(3) = \{2, 3, 1, 4\} = B_6$



(d) $N_{k-1}^{(1,2)}(3) = \{3, 1, 4\} = B_7$

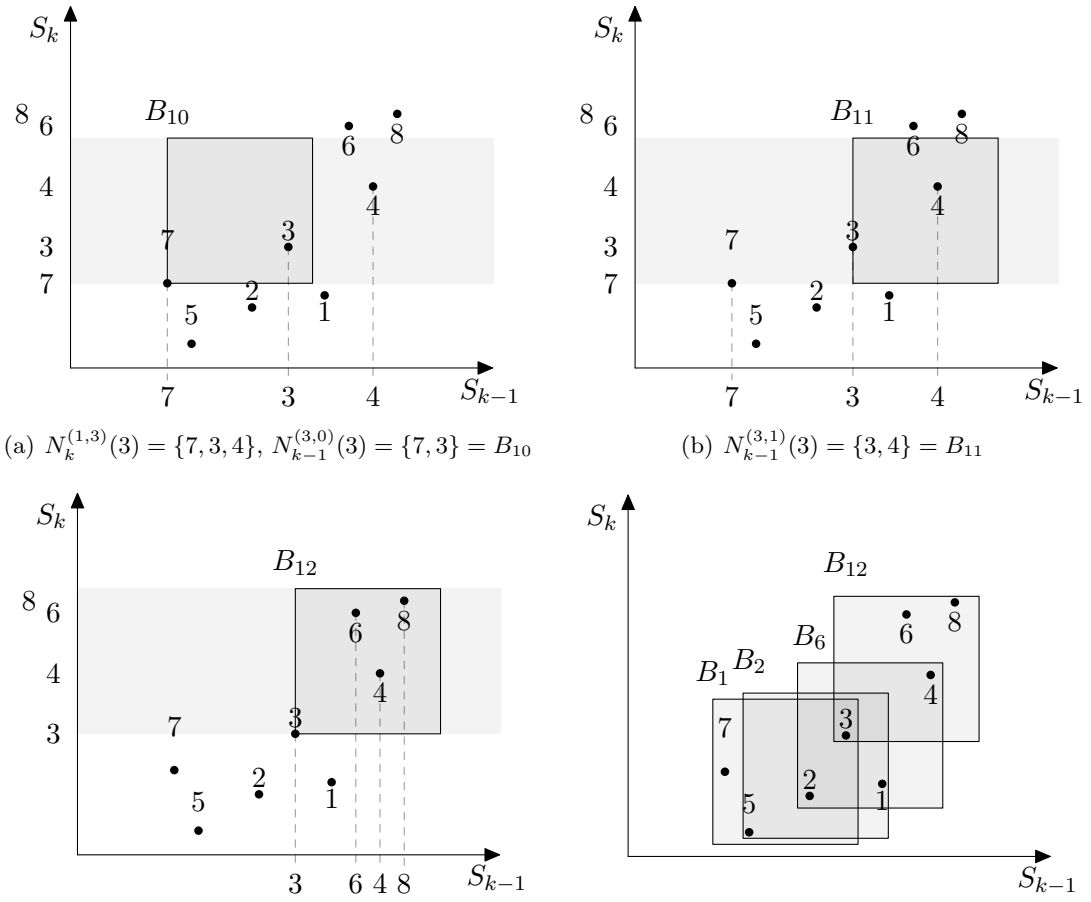


(e) $N_k^{(1,2)}(3) = \{1, 7, 3, 4\}$, $N_{k-1}^{(2,0)}(3) = \{7, 3\} = B_8$



(f) $N_{k-1}^{(2,1)}(3) = \{3, 1, 4\} = B_9$

FIGURE 4.3 – Exécution de l'algorithme 2 sur un ensemble de 8 entités.



(a) $N_k^{(1,3)}(3) = \{7, 3, 4\}$, $N_{k-1}^{(3,0)}(3) = \{7, 3\} = B_{10}$

(b) $N_{k-1}^{(3,1)}(3) = \{3, 4\} = B_{11}$

(c) $N_k^{(1,4)}(3) = \{3, 6, 4, 8\}$, $N_{k-1}^{(4,0)}(3) = \{3, 6, 4, 8\} = B_{12}$

(d) $\mathcal{M}(3) = \{B_1, B_2, B_6, B_{12}\}$.

FIGURE 4.4 – Exécution de l'algorithme 2 sur un ensemble de 8 entités.

L'algorithme commence par ordonner $N(3)$ *en fonction de la* i*-ème coordonnée des entités à l'instant* k*. On a* $N_k^{(1)}(3) = \{5, 2, 1, 7, 3, 4, 6, 8\}$ *(voir la figure 4.2(b)). L'entité* 3 *sélectionne ensuite parmi* $N_k^{(1)}(3)$ *l'ensemble des entités dont la distance les séparant de la première entité (entité* 5*) de* $N_k^{(1)}(3)$ *n'excède pas* $2r$ *(voir la figure 4.2(c)). On a :*

$$\begin{aligned} N_k^{(1,0)}(3) &= \{\ell \in N_k^{(1)}(3) \mid \|p_{k-1}(\ell) - p_{k-1}(5)\| \leq 2r \text{ et } \|p_k(\ell) - p_k(5)\| \leq 2r\} \\ &= \{5, 2, 1, 7, 3\}. \end{aligned}$$

L'algorithme est appelé récursivement avec :

$$N(3) = N_k^{(1,0)}(3), i = 1, \ell = 0 \text{ et } \mathcal{M}_k(3) = \emptyset.$$

Toutes les dimensions ($d = 1$) *ayant été parcourues à l'instant* k*, l'algorithme considère les entités de* $N(3)$ *suivant leurs positions à l'instant* $k-1$*. On a* $N_{k-1}^{(0)}(3) = \{7, 5, 2, 3, 1\}$*. Toutes les entités dont la distance les séparant de la première (entité* 7*) n'excède pas* $2r$ *sont sélectionnées (voir figure 4.2(d)). On a :*

$$\begin{aligned} N_{k-1}^{(0,0)}(3) &= \{\ell \in N_{k-1}^{(1)}(3) \mid \|p_{k-1}(\ell) - p_{k-1}(7)\| \leq 2r \text{ et } \|p_k(\ell) - p_k(7)\| \leq 2r\} \\ &= \{7, 5, 2, 3\} = B_1. \end{aligned}$$

On a $N_{k-1}^{(1,1)}(3) = \{7, 5, 2, 3\} = B_1$*. Toutes les dimensions aux deux instants ont été parcourues et* $\mathcal{M}_k(3) = \emptyset$*. On a donc* $\forall B \in \mathcal{M}_k(3), B \nsubseteq B_1, B_1 \nsubseteq B$*.* B_1 *a donc un mouvement* r*-cohérent maximal. On ajoute* B_1 *à* $\mathcal{M}_k(3)$ *:* $\mathcal{M}_k(3) = \{B_1\}$*. La récursivité s'achève.*

L'algorithme supprime l'entité 7 *de l'ensemble* $N_{k-1}^{(0)}(3)$*. On a alors* $N_{k-1}^{(0)}(3) = \{5, 2, 3, 1\}$ *(voir figure 4.2(e)). En appliquant les mêmes étapes que précédemment, nous obtenons :*

$$N_{k-1}^{(0,1)}(3) = \{5, 2, 3, 1\} = B_2.$$

On a pour tout $B \in \mathcal{M}_k(3), B \nsubseteq B_1, B_1 \nsubseteq B$*.* B_2 *a donc un mouvement* r*-cohérent maximal, il est ajouté à* $\mathcal{M}_k(3)$*. On a* $\mathcal{M}_k(3) = \{B_1, B_2\}$*. La récursivité s'achève.*

L'algorithme supprime l'entité 5 *de l'ensemble* $N_{k-1}^{(0)}(3)$ *ce qui conduit à* $N_{k-1}^{(0)}(3) = \{2, 3, 1\}$ *(voir figure 4.2(f)). L'algorithme est appliqué sur cet ensemble. On a :*

$$N_{k-1}^{(0,2)}(3) = \{5, 2, 3, 1\} = B_3.$$

On a $B_3 \subseteq B_2 \in \mathcal{M}_k(3)$*, donc* B_3 *n'est pas maximal, il n'est pas ajouté à* $\mathcal{M}_k(3)$*. L'algorithme supprime l'entité* 2 *de l'ensemble* $N_{k-1}^{(0)}(3)$*, on a* $N_{k-1}^{(0)}(3) = \{3, 1\} = B_4$ *(voir figure 4.3(a)). De la même manière,* $B_4 \subseteq B_2 \in \mathcal{M}_k(3)$*,* B_4 *n'est donc pas maximal.*

L'algorithme supprime l'entité 3 *de l'ensemble* $N_{k-1}^{(0)}(3)$*. Tous les sous-ensembles de* $N_k^{(1,0)}(3)$ *contenant l'entité* 3 *ont été parcourus. La récursivité s'achève pour cet ensemble.*

L'algorithme supprime la première entité de l'ensemble $N_k^{(1)}(3)$*, et sélectionne toutes les entités dont la distance les séparant de la première (entité* 2*) n'excède pas* $2r$ *(voir figure 4.3(b)), on a :*

$$N_k^{(1,1)}(3) = \{2, 1, 7, 3, 4\}.$$

L'algorithme applique la même procédure de manière récursive sur ce nouvel ensemble. Il teste $B_5 = \{7, 2, 3\} \subseteq B_1$ *(figure 4.3(c)).*

L'algorithme continue ainsi jusqu'à avoir testé tous les ensembles contenant 3 *ayant un mouvement r-cohérent (figures 4.3(d)- 4.4(c)). À la fin de l'exécution de cet algorithme, on a* $\mathcal{M}_k(3) = \{B_1, B_2, B_6, B_{12}\}$*. Ces ensembles sont illustrés dans la figure 4.4(d).*

Une approche centralisée de cet algorithme teste dans tous les cas $\mathcal{O}(|A_k|^{2d})$ ensembles afin de déterminer pour chaque entité $j \in A_k$, les ensembles $\overline{\mathcal{W}}_k(j)$. À l'inverse, chaque entité supervisée exécutant l'algorithme 2 teste $\mathcal{O}(|N(j)|^{2d})$ ensembles afin de déterminer l'ensemble $\overline{\mathcal{W}}_k(j)$. Ainsi dans le pire des cas, si tous les éléments de A_k sont concentrés dans le voisinage de j, cette entité teste alors $\mathcal{O}(|A_k|^{2d})$ ensembles.

4.1.2 Caractérisation des fautes

L'algorithme 3 présente le pseudo-code exécuté par l'entité supervisée j afin de déterminer le type de faute ayant provoqué la défaillance perçue par j. L'algorithme commence par calculer les ensembles contenant j ayant un mouvement τ-dense maximal. Si aucun ensemble n'est trouvé, le théorème 2 s'applique (lignes 2-5).

Dans le cas contraire, les ensembles $J_k(j)$ et $L_k(j)$ sont construits (lignes 7-19). L'algorithme cherche alors à appliquer le corollaire 2 en cherchant parmi les ensembles contenant j ayant un mouvement τ-dense maximal, si l'un d'eux a une intersection avec $J_k(j)$ dont le cardinal est supérieur à τ.

Ce corollaire est plus efficace en terme d'implémentation que le théorème 3. En effet, on ne teste que $|\overline{\mathcal{W}}_k(j)|$ ensembles dans ce cas, tandis que la condition décrite dans le théorème 3 nécessite pour chaque ensemble B de $\overline{\mathcal{W}}_k(j)$ de tester tous les sous-ensembles de B de cardinal strictement supérieur à τ.

L'algorithme retourne alors `massive` dès qu'un ensemble satisfait cette condition (ligne 21). Si aucun ensemble satisfaisant cette condition n'est trouvé, l'algorithme retourne un état indéterminé (ligne 28).

Comme nous l'avons vu dans la section 3.3.2, le théorème 3 n'est qu'une condition suffisante pour caractériser l'appartenance d'une entité à M_k. Aussi, dans certains cas, il arrive que l'algorithme 3 retourne un état indéterminé pour une entité j alors que celle-ci appartient effectivement à M_k. Il faut donc utiliser le théorème 4 afin éviter ce faux négatif.

La mise en œuvre du théorème 4 est faite dans l'algorithme 5. D'un point de vue algorithmique, cela revient à remplacer la ligne 28 de l'algorithme 3 par la mise en œuvre du théorème 4 et du corollaire 3 (lignes 23-35 de l'algorithme 5).

L'algorithme itère sur les entités de $L_k(j)$ afin de créer avec l'algorithme 6 une collection dans laquelle j appartient à un ensemble ayant un mouvement non τ-dense. Si

Algorithme 2 : j.maxMotions($N(j)$, i, ℓ, $\mathcal{M}_k(j)$)

Données :
$N(j)$: ensemble des entités de A_k dont la distance les séparant de j n'excède pas $2r$ aux instants $k-1$ et k,
$\mathcal{M}_k(j)$: famille des ensembles contenant j ayant un mouvement r-cohérent maximal,
i : dimension courante,
$\ell = 0$ état courant du système,
$\ell = 1$ état précédent du système.
Sorties : $\mathcal{M}_k(j)$

début
  si $i > d$ et $\ell = 0$ **alors**
    $\ell \leftarrow \ell + 1$;
    $i \leftarrow 0$;
  fin
  $i \leftarrow i + 1$;
  si $N(j) \notin \mathcal{M}_k(j)$ et $i \leq d$ et $\ell \leq 1$ **alors**
    $x_{set} \leftarrow \{q_{i,k-\ell}(x) \mid x \in N(j)\}$;
    $x_m \leftarrow \min(x_{set})$;
    $N \leftarrow \{x \in N(j) \mid x_m \leq q_{i,k-\ell}(x) \leq x_m + 2r\}$;
    tant que $x_m < q_{i,k-\ell}(j)$ et $N \neq \emptyset$ **faire**
      $\mathcal{M}_k(j) \leftarrow$`j.maxMotions(`$N,i,\ell,\mathcal{M}_k(j)$`)`;
      $x_{set} \leftarrow x_{set} \setminus \{x_m\}$;
      $x_m \leftarrow \min(x_{set})$;
      $N \leftarrow \{x \in N(j) \mid x_m \leq q_{i,k-\ell}(x) \leq x_m + 2r\}$;
    fin
  sinon si $\forall M \in \mathcal{M}_k(j), N(j) \not\subseteq M$ **alors**
    $\mathcal{M}_k(j) \leftarrow \{M \in \mathcal{M}_k(j) \mid M \not\subseteq N(j)\}$;
    $\mathcal{M}_k(j) \leftarrow \mathcal{M}_k(j) \cup \{N(j)\}$;
  fin
  retourner $\mathcal{M}_k(j)$;
fin

une telle collection est exhibée alors j est dans un état indéterminé, sinon, le théorème 4 s'applique et garantit que l'entité j a perçu une défaillance due à une faute massive.

L'algorithme 6 construit récursivement une famille d'ensembles ayant des mouvements τ-denses respectant les hypothèses du corollaire 3. Il fonctionne de manière similaire à l'algorithme 1.

En effet, celui-ci construit de manière récursive une famille dans laquelle les éléments ℓ de $L_k(j)$ appartiennent à des ensembles ayant des mouvements τ-dense. L'algorithme est initialisé avec les ensembles S et L suivants :

$$S = \left(\bigcup_{\ell \in L_k(j)} D_k(\ell) \right) \setminus \{j\},$$
$$L = L_k(j).$$

De manière similaire à l'algorithme 1, l'algorithme 6 sélectionne une entité pivot $\ell \in L$ et un ensemble C composé d'entités de S tel que $\ell \in C$ et C a un mouvement τ-dense maximal parmi les entités de S. Cet ensemble C est alors ajouté à la famille $\mathcal{F}$ courante et ses entités retirées de S. L'algorithme s'applique de nouveau sur les éléments restants de S et L afin de compléter la collection en cours $\mathcal{F}$.

La récursivité de l'algorithme 6 s'arrête quand il n'est plus possible d'adjoindre à $\mathcal{F}$ de nouveaux ensembles C. On teste alors parmi les entités restantes, s'il est possible d'exhiber un ensemble contenant l'entité j ayant un mouvement τ-dense. Si ce n'est pas possible, on a alors construit une famille $\mathcal{F}$ vérifiant le corollaire 3 et donc l'entité j est en état indéterminé. Dans le cas contraire, il faut tester d'autres collections.

À la différence de l'algorithme 1 qui construit une partition de l'ensemble d'entités S, l'algorithme 6 cherche à construire une famille respectant les hypothèses du corollaire 3. Ainsi, lorsque l'algorithme s'arrête et que la collection ne vérifie pas ces hypothèses, l'algorithme remplace le dernier ensemble C ajouté de taille s par un autre ensemble de taille $s-1$ et calcule ainsi une nouvelle collection. Lorsque tous les sous-ensembles de C de taille strictement supérieure à τ ont été testés, la récursivité s'arrête. Au niveau supérieur, on remplace alors l'avant dernier ensemble de taille s' de la famille $\mathcal{F}$ par un sous-ensemble de taille $s'-1$. L'algorithme est alors réexécuté récursivement sur cette nouvelle famille. Ainsi, toutes les collections d'ensembles d'entités de $L_k(j)$ ayant des mouvement τ-denses sont calculées si $j \in M_k$.

Cependant, l'algorithme tel qu'il est décrit dans le pseudo-code 6 peut générer plusieurs fois la même collection, augmentant ainsi le coût de calcul de la caractérisation de fautes. Afin d'interdire ces exécutions induisant le même résultat et dans un soucis de lisibilité, les lignes 3 et 5 diffèrent du pseudo-code dans leur implémentation. On remplace alors :

$$M \cap S \setminus \bigcup_{C_i \in \mathcal{F}} C_i \text{ par } \{m \in M \cap S \setminus \bigcup_{C_i \in \mathcal{F}} C_i \mid \forall m \in L, m > \ell\}$$

De même, l'ensemble $B \in \{B \subseteq (M \setminus \{j\}) \cap S \mid |B| = s\}$ est remplacé par

$$\{B \subseteq \{m \in (M \setminus \{j\}) \cap S \setminus \bigcup_{C_i \in \mathcal{F}} C_i \mid |B| = s \text{ et } \forall m \in L, m > \ell\}.$$

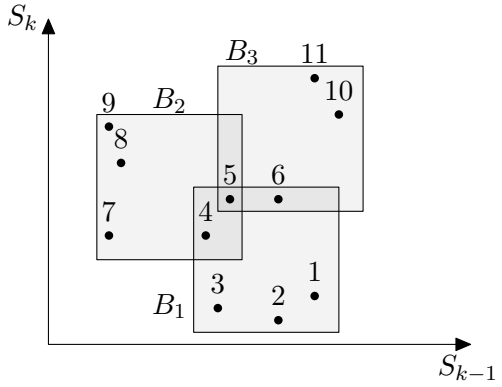


FIGURE 4.5 – Exécution de l'algorithme 6 sur un ensemble de 11 entités.

De cette manière, on utilise l'ordre naturel présent sur les identifiants pour ne tester qu'une fois chaque ensemble. En effet, si on considère l'ensemble $B = \{b_1, \ldots, b_v\} \subseteq [\![1, n]\!]$, celui-ci ne sera testé pour être ajouté à la famille $\mathcal{F}$ qu'au moment où b_1 sera choisit comme pivot pour l'exécution de l'algorithme 6.

Exemple 14 *La figure 4.5 illustre le déroulement de l'algorithme 6. On considère un système composé de* 11 *entités consommant un unique service (*i.e.*, $d = 1$). De plus, on considère que $\tau = 3$ et que toutes les entités du système perçoivent une défaillance.*

Considérons l'entité 1*. On a $\overline{\mathcal{W}}_k(1) = \{1, 2, 3, 4, 5, 6\} \neq \emptyset$, donc $1 \notin I_k$. On a $D_k(1) = \{1, 2, 3, 4, 5, 6\}$. Les entités $1, 2, 3$ n'appartiennent qu'à un seul ensemble B_1 ayant un mouvement r-cohérent τ-dense. On a $1, 2, 3 \in J_k(1)$.*

L'ensemble $B_2 = \{4, 5, 7, 8, 9\}$ a un mouvement τ-dense et $1 \notin B_2$. On a alors $4 \in L_k(1)$. De la même manière, l'ensemble $B_3 = \{5, 6, 10, 11\}$ a un mouvement τ-dense et $1 \notin B_3$. On a $5, 6 \in L_k(1)$ et donc $L_k(1) = \{4, 5, 6\}$ et $J_k(1) = \{1, 2, 3\}$. $L_k(1) = \{3, 4, 5\}$. On a donc $|J_k(1)| = 3 <= \tau$, le corollaire 2 ne peut donc pas s'appliquer.

On cherche à déterminer si l'entité 1 *est en état indéterminé ($1 \in U_k$) ou si elle appartient à M_k. L'algorithme 5 construit les ensembles $S = \left(\bigcup_{\ell \in L} D_k(\ell)\right) \setminus \{1\} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, $L = \{4, 5, 6\}$ et $\mathcal{F} = \{\}$. L'algorithme 6 sélectionne initialement l'entité* 3 *comme pivot. On a $\overline{\mathcal{W}}_k(3) = \{B_1, B_2\}$. On ajoute alors $C_1 = B_1 \setminus \{1\}$ à la famille $\mathcal{F}$, on a $\mathcal{F} = \{C_1\}$.*

L'algorithme est appelé récursivement sur l'ensemble $S = \{7, 8, 9, 10, 11\}$, avec $L = \{5, 6\}$ et $\mathcal{F} = \{C_1\}$. Comme le montre la figure 4.6, il n'existe pas de sous-ensemble de S ayant un mouvement r-cohérent τ-dense, la récursivité s'achève. On teste alors les conditions du corollaire 3. On a $C_1 \cup \{1\} = B_1 \in \overline{\mathcal{W}}_k(1)$, le deuxième terme de la condition du corollaire 3 n'est donc pas respecté.

L'algorithme cherche à construire une nouvelle famille $\mathcal{F}$ en supprimant C_1 de $\mathcal{F}$ et en ajoutant $C_2 = B_2$ à celle-ci. L'algorithme est appelé récursivement sur l'ensemble $S = \{2, 3, 6, 10, 11\}$ $L = \{5, 6\}$ et $\mathcal{F} = \{C_2\}$. Comme le montre la figure 4.7, il n'existe

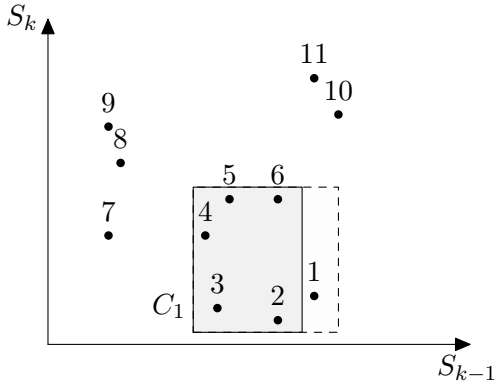


FIGURE 4.6 – Exécution de l'algorithme 6 sur un ensemble de 11 entités.

pas de sous-ensemble de S ayant un mouvement r-cohérent τ-dense, la récursivité s'achève.

On teste les conditions du corollaire 3. On considère l'ensemble $B = \{1, 2, 3, 6\}$. On a $B \subset B_1$, et $|B| = 4 > \tau$ donc on a $B \in \mathcal{W}_k(1)$. De plus, on a $D_k(1) \setminus \bigcup_{C \in \mathcal{F}} C = D_k(1) \setminus C_2 = \{1, 2, 3, 6, 10, 11\}$. On a donc $B \subseteq D_k(1) \setminus \bigcup_{C \in \mathcal{F}} C$. Le premier terme de la condition du corollaire 3 n'est donc pas respecté.

L'algorithme cherche à construire une nouvelle famille $\mathcal{F}$, on supprime C_1 de $\mathcal{F}$. Tous les ensembles de $\overline{\mathcal{W}}_k(3)$ ont été testés sans succès. L'algorithme teste alors successivement les sous-ensembles de chaque ensemble de $\overline{\mathcal{W}}_k(3)$. Aucun des sous-ensembles de C_1 ne permet de construire une famille satisfaisant les conditions du corollaire 3 pour les mêmes raisons que C_1. L'algorithme teste ensuite les sous-ensembles de C_2.

L'algorithme ajoute ensuite l'ensemble $C_3 = \{4, 7, 8, 9\}$ à $\mathcal{F} = \{\}$. L'algorithme est appelé récursivement sur l'ensemble $S = \{2, 3, 5, 6, 10, 11\}$ $L = \{5, 6\}$ et $\mathcal{F} = \{C_3\}$. Celui-ci sélectionne l'entité 5 comme pivot et ajoute l'ensemble $C_4 = \{5, 6, 10, 11\} \in \overline{\mathcal{W}}_k(5)$ à $\mathcal{F}$. On a $\mathcal{F} = \{C_3, C_4\}$ comme illustré sur la figure 4.8. De plus, on a $L = \emptyset$, la récursivité s'achève.

On a $D_k(1) \setminus \bigcup_{C \in \mathcal{F}} C = \{1, 2, 3\}$. On a $|\{1, 2, 3\}| = 3 \leq \tau$, donc $\forall B \in \mathcal{W}_k(1)$ on a $B \nsubseteq \{1, 2, 3\}$. De plus, $C_3 \cup \{1\}$ et $C_4 \cup \{1\}$ ne sont pas r-cohérents, donc $C_3 \cup \{1\} \notin \mathcal{W}_k(1)$ et $C_4 \cup \{1\} \notin \mathcal{W}_k(1)$. Le corollaire 3 est vérifié, l'entité 1 est en état indéterminé.

4.2 Évaluation

Nous évaluons dans cette section ce modèle ainsi que les algorithmes proposés. Du fait de l'absence de vérité terrain comprenant à la fois des mesures de bout-en-bout collectées auprès d'utilisateurs et de traces effectives de défaillances permettant de labelliser chaque variation anormale de mesure, cette évaluation est menée par simulation. L'idée consiste à générer une vérité synthétique que l'on maîtrise afin de pouvoir comparer les résultats fournis par les algorithmes présentés à la vérité synthétique générée. Durant ces simulations, quatre métriques ont été analysées :

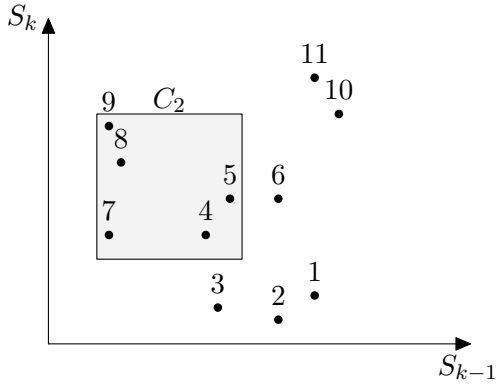


FIGURE 4.7 – Exécution de l'algorithme 6 sur un ensemble de 11 entités.

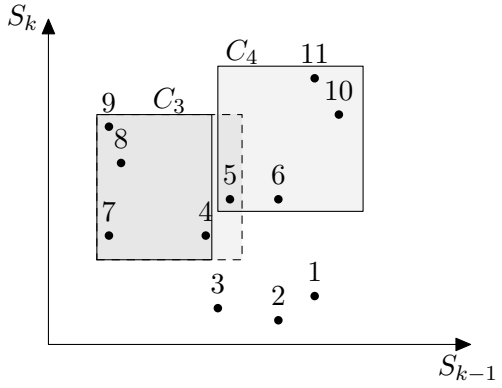


FIGURE 4.8 – Exécution de l'algorithme 6 sur un ensemble de 11 entités.

Algorithme 3 : j.characterize()

Données :
$N(j) = N_k(j) \cap N_{k-1}(j)$: ensemble des entités dont la distance les séparant de j aux instants $k-1$ et k n'excède pas $2r$,
τ : seuil de densité.
Sorties :
Type de faute dont la défaillance est perçue par j : mise en œuvre du Théorème 2 et du Corollaire 2

```
1  début
2      M_k(j) ← j.maxMotions(N(j),0,0,{});
3      W̄_k(j) ← {M ∈ M_k(j) | |M| > τ} ;
4      si W̄_k(j) = ∅ alors
5          faulttype← Isolated;
6      sinon
7          J_k(j) ← ∅;
8          L_k(j) ← ∅;
9          pour chaque M ∈ W̄_k(j) faire
10             pour chaque ℓ ∈ M faire
11                 M_k(ℓ) ← ℓ.maxMotions(N(ℓ),0,0,{});
12                 W̄_k(ℓ) ← {M' ∈ M_k(ℓ) | |M'| > τ};
13                 si ∃M' ∈ W̄_k(ℓ) tel que j ∉ M' alors
14                     L_k(j) ← L_k(j) ∪ {ℓ};
15                 sinon
16                     J_k(j) ← J_k(j) ∪ {ℓ};
17                 fin
18             fin
19         fin
20         si ∃M ∈ W̄_k(j) tel que |M ∩ J_k(j)| > τ alors
21             faulttype ← Massive;
22         sinon
23             faulttype ← Unresolved;
24         fin
25     fin
26     retourner faulttype;
27 fin
```

Algorithme 4 : j.characterize()

Données :
$N(j) = N_k(j) \cap N_{k-1}(j)$: ensemble des entités dont la distance les séparant de j aux instants $k-1$ et k n'excède pas $2r$,
τ : seuil de densité.
Sorties :
Type de faute dont la défaillance est perçue par j : mise en œuvre du Théorème 2, du Corollaire 2 et du Théorème 6

début
  $\mathcal{M}_k(j) \leftarrow$ j.maxMotions($N(j)$,0,0,{});
  $\overline{\mathcal{W}}_k(j) \leftarrow \{M \in \mathcal{M}_k(j) \mid |M| > \tau\}$;
  si $\overline{\mathcal{W}}_k(j) = \emptyset$ **alors**
    faulttype← Isolated;
  sinon
    $J_k(j) \leftarrow \emptyset$;
    $L_k(j) \leftarrow \emptyset$;
    $E_k(j) \leftarrow \emptyset$;
    pour chaque $M \in \overline{\mathcal{W}}_k(j)$ **faire**
      pour chaque $\ell \in M$ **faire**
        $\mathcal{M}_k(\ell) \leftarrow \ell$.maxMotions($N(\ell)$,0,0,{});
        $\overline{\mathcal{W}}_k(\ell) \leftarrow \{M' \in \mathcal{M}_k(\ell) \mid |M'| > \tau\}$;
        si $\overline{\mathcal{W}}_k(\ell) = \overline{\mathcal{W}}_k(j)$ **alors**
          $E_k(j) \leftarrow E_k(j) \cup \{\ell\}$;
        fin
        si $\exists M' \in \overline{\mathcal{W}}_k(\ell)$ tel que $j \notin M'$ **alors**
          $L_k(j) \leftarrow L_k(j) \cup \{\ell\}$;
        sinon
          $J_k(j) \leftarrow J_k(j) \cup \{\ell\}$;
        fin
      fin
    fin
    si $j = \min\{\ell \in E_k(j)\}$ **alors**
      si $\exists M \in \overline{\mathcal{W}}_k(j)$ tel que $|M \cap J_k(j)| > \tau$ **alors**
        faulttype ← Massive;
      sinon
        faulttype ← Unresolved;
      fin
      pour chaque $\ell \in E_k(j)$ **faire**
        envoyer caractérisation faulttype **à** ℓ;
      fin
    sinon
      attendre caractérisation de $\min\{\ell \in E_k(j)\}$;
    fin
  fin
  retourner faulttype;
fin

Algorithme 5 : j.characterize()

Données : $N(j) = N_k(j) \cap N_{k-1}(j)$: ensemble des entités dont la distance les séparant de j aux instants $k-1$ et k n'excède pas $2r$, τ : seuil de densité.

Sorties : Type de faute dont la défaillance est perçue par j : mise en œuvre du Théorème 2, du Corollaire 2, du Théorème 4 et du Théorème 3

début
  $\mathcal{M}_k(j) \leftarrow$ j.`maxMotions`($N(j)$,0,0,{});
  $\overline{\mathcal{W}}_k(j) \leftarrow \{M \in \mathcal{M}_k(j) \mid |M| > \tau\}$;
  si $\overline{\mathcal{W}}_k(j) = \emptyset$ **alors**
    faulttype← Isolated;
  sinon
    $J_k(j) \leftarrow \emptyset$;
    $L_k(j) \leftarrow \emptyset$;
    pour chaque $M \in \overline{\mathcal{W}}_k(j)$ **faire**
      pour chaque $\ell \in M$ **faire**
        $\mathcal{M}_k(\ell) \leftarrow \ell$.`maxMotions`($N(\ell)$,0,0,{}) ;
        $\overline{\mathcal{W}}_k(\ell) \leftarrow \{M' \in \mathcal{M}_k(\ell) \mid |M'| > \tau\}$;
        si $\exists M' \in \overline{\mathcal{W}}_k(\ell)$ tel que $j \notin M'$ **alors**
          $L_k(j) \leftarrow L_k(j) \cup \{\ell\}$;
        sinon
          $J_k(j) \leftarrow J_k(j) \cup \{\ell\}$;
        fin
      fin
    fin
    si $\exists M \in \overline{\mathcal{W}}_k(j)$ tel que $|M \cap J_k(j)| > \tau$ **alors**
      faulttype← Massive;
    sinon
      $S \leftarrow \bigcup_{\ell \in L} D_k(\ell)$;
      $S \leftarrow S \setminus \{j\}$;
      $\mathcal{F} \leftarrow \{\}$;
      $R \leftarrow L_k(j)$;
      tant que $R \neq \emptyset$ et $\neg$ j.`check`($\mathcal{F}$) **faire**
        Choisir $\ell \in R$;
        $R \leftarrow R \setminus \{\ell\}$;
        $\mathcal{F} \leftarrow$j.`isolate`(S,R,$\mathcal{F}$,ℓ);
      fin
      si j.`check`($\mathcal{F}$) **alors**
        faulttype← Unresolved;
      sinon
        faulttype← Massive;
      fin
    fin
  fin
  retourner faulttype;
fin

Algorithme 6 : j.isolate(S,L,$\mathcal{F}$,ℓ)

Données : S : ensemble des entités appartenant à un ensemble ayant un mouvement τ-dense impliquant une entité de $L_k(j)$
Sorties : Une famille $\mathcal{F}$ satisfaisant le Corollaire 3

```
1  début
2      pour chaque M ∈ W_k(ℓ) faire
3          s ← |M ∩ S \ ⋃_{B_i∈F} B_i|;
4          tant que s > τ faire
5              pour chaque B ∈ {B ⊆ M ∩ S | |B| = s et j ∉ B} faire
6                  F ← F ∪ {B};
7                  S ← S \ B;
8                  Choisir m ∈ L;
9                  si L \ {m} ≠ ∅ alors
10                     F ←j.isolate(S,L \ {m},F,m);
11                 fin
12                 si j.check(F) alors
13                     retourner F;
14                 fin
15             fin
16             s ← s − 1;
17         fin
18     fin
19     retourner F;
20 fin
```

— le coût moyen en termes d'ensembles calculés et testés,
— le taux d'erreur de l'algorithme 3 par rapport à l'algorithme 5
— la sensibilité de cette approche par rapport à la fréquence d'échantillonnage des états du système,
— l'erreur commise par les algorithmes sur une trace synthétique ne respectant pas les hypothèses du modèle.

Dans notre approche, les entités effectuent régulièrement des mesures de performances des services consommés. De cette manière, la fréquence des mesures peut-être réglée localement en fonction de la variation anormale de ces mesures. Suite à la détection de variation de qualité, l'entité se déplace dans l'espace des qualités E. Il est donc suffisant de connaître les entités présentes à cette nouvelle position pour déterminer la présence éventuelle d'autres entités ayant perçu la même défaillance. Ce type d'approche signifie donc qu'il n'est pas nécessaire que les entités effectuent leurs mesures de manière totalement synchronisée. Il est en effet suffisant pour une entité percevant une défaillance d'être capable de déterminer les éventuelles autres entités ayant perçu cette défaillance. La mise en œuvre de cette recherche sera décrite dans le chapitre 5.

4.2.1 Paramètres de simulation

Diverses configurations de paramètres ont été testées. On considère un ensemble S de $n = 1000$ entités supervisées consommant $d = 2$ services. Les entités se positionnent ainsi dans un espace des qualités E à deux dimensions. Initialement, les entités sont distribuées uniformément sur E définissant ainsi l'état initial S_0 du système.

Dans ce modèle, on considère comme isolée une faute impactant un nombre d'entités inférieur au seuil de densité τ. Par ailleurs, la restriction **R2** du modèle suppose qu'une même faute a un impact similaire sur chacune des entités en percevant la défaillance. On suppose alors ces fautes indépendantes.

Une partition d'anomalies contenant $A \in [\![1, 80]\!]$ ensembles est générée. Ce nombre d'ensembles A est un paramètre de la simulation. Les anomalies (au sens de la définition 9) sont générées en choisissant uniformément une entité $j \in S$.

Pour chaque entité j ainsi choisie, on détermine aléatoirement un nombre t d'entités à déplacer parmi celles présentes dans la boule de rayon r centrée sur j et telles que $a_k(\ell) =$`FAUX`. Le nombre d'entités t est tel que $t \leq \tau$ avec probabilité G et $t > \tau$ avec probabilité $1 - G$.

On choisit ensuite uniformément t entités présentes dans le voisinage de l'entité j. Le paramètre G nous permet ainsi de déterminer la proportion d'anomalies isolées et d'anomalies massives que l'on souhaite obtenir dans la vérité synthétique générée. Les valeurs extrêmes de G correspondent à des cas idéaux. Dans le cas $G = 0$, seules des anomalies massives sont générées. Cela correspond donc à un système dans lequel les entités supervisées sont parfaites et ne subissent aucune faute. À l'inverse, dans le cas $G = 1$, seules des anomalies isolées sont générées. Cela signifie alors que seules les entités supervisées peuvent subir des fautes et que le reste du système est parfait et ne subit aucune faute. Les entités ainsi choisies sont ensuite déplacées dans E de

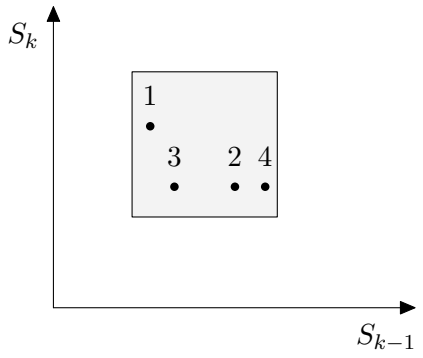


FIGURE 4.9 – Génération de fautes isolées menant à une faute massive.

manière similaire, c'est à dire que l'ensemble a un mouvement r-cohérent. Pour chaque entité ℓ déplacée, on positionne la valeur de $a_k(\ell)$ à `VRAI`. Cette manière de générer les anomalies dans le système peut amener plusieurs entités ayant chacune subi une faute isolée à être suffisamment proches pour appartenir à un même mouvement τ-dense. Dans ce cas, la partition générée ne respecte pas la condition C2 de la définition 8 de partition d'anomalies. La partition est alors modifiée en regroupant toutes ces entités au sein d'une anomalie massive et celles-ci sont alors marquées comme ayant été impactées par une faute massive.

La figure 4.9 illustre ce cas. On a généré quatre anomalies isolées et celles-ci sont suffisamment proches les unes des autres pour que le modèle les considère comme une anomalie massive (restriction **R3**). On modifie alors la vérité synthétique générée pour marquer ces entités comme ayant subi une faute massive. On compare ensuite la vérité synthétique ainsi générée aux résultats fournis par simulation.

Le rayon de cohérence r et le seuil de densité τ sont des paramètres fondamentaux dans notre approche : le premier caractérise la forte corrélation entre les variations de mesure perçues par les entités supervisées lors de l'occurrence d'une défaillance tandis que le second permet de discriminer les fautes isolées des fautes massives. Ces paramètres r et τ sont dimensionnés de sorte que la probabilité que plus de τ fautes isolées apparaissent dans le voisinage d'une entité j soit négligeable.

Soient $N_r(j)$ la variable aléatoire représentant le nombre d'entités dont la distance à j à n'excède pas $2r$ et $F_r(j)$ la variable aléatoire représentant le nombre d'entités impactées par une faute isolée dont la distance à j n'excède pas $2r$. On a alors

$$\Pr\{N_r(j) = i\} = \binom{n-1}{i} q_j^i (1-q_j)^{n-1-i}$$

avec q_j la probabilité que l'entité ℓ soit dans le voisinage de l'entité j.

Étant donné la position $p_k(j)$ de l'entité j à l'instant k, le voisinage $V_k(j) \subset E$ de l'entité j à l'instant k est défini par l'ensemble suivant :

$$V_k(j) = \{\ell \in S \mid \|p_k(\ell) - p_k(j)\| \leq 2r\}, \mid V_k(j) \mid = N_r(j).$$

On a :

$$\begin{aligned}\Pr\{F_r(j) > \tau\} &= 1 - \sum_{\ell=0}^{\tau} \Pr\{F_r(j) = \ell\} \\ &= 1 - \sum_{m=0}^{n-1} \sum_{\ell=0}^{\tau \wedge m} \Pr\{F_r(j) = \ell \mid N_r(j) = m\} \Pr\{N_r(j) = m\} \\ &= 1 - \sum_{m=0}^{n-1} \sum_{\ell=0}^{\tau \wedge m} \binom{m}{\ell} f^{\ell}(1-f)^{m-\ell} \Pr\{N_r(j) = m\}.\end{aligned}$$

où $x \wedge y = \min(x, y)$ et f désigne la probabilité qu'une entité subisse une faute isolée dans l'intervalle de temps $[k-1, k]$.

On a donc :

$$\Pr\{F_r(j) > \tau\} = 1 - \sum_{m=0}^{n-1} \sum_{\ell=0}^{\tau \wedge m} \binom{m}{\ell} f^{\ell}(1-f)^{m-\ell} \binom{n-1}{m} q_j^m (1-q_j)^{n-1-m}.$$

On cherche à fixer r et τ de sorte que la probabilité d'avoir plus de τ fautes indépendantes perçues par des entités dans le voisinage de j soit négligeable. Étant donné $\varepsilon > 0$ aussi petit que l'on veut, on cherche r et τ vérifiant la relation suivante :

$$\Pr\{F_r(j) \leq \tau\} < 1 - \varepsilon.$$

La figure 4.10 représente la fonction de répartition de la variable aléatoire $N_r(j)$ en fonction de la taille m du voisinage de l'entité j dans E pour différentes valeurs de rayon de cohérence r. Cette courbe illustre clairement l'impact de r sur le nombre d'entités dans le voisinage de j. Naturellement, plus la taille du système augmente plus le nombre d'entités dans le voisinage de j augmente. On dimensionne r de sorte que la taille du voisinage ne soit pas trop importante. Dans le cadre des simulations effectuées, pour $n = 1000$, on fixe $r = 0.03$ assurant un nombre de voisins raisonnable et ainsi le passage à l'échelle de notre approche.

La figure 4.11 illustre quant à elle la probabilité d'avoir moins de τ fautes isolées dans ce voisinage. De la même manière, il est naturel de constater que cette probabilité décroît lorsque la taille du système augmente. En effet, plus la taille du système augmente plus la probabilité de constater un nombre de fautes isolées dans le voisinage augmente, et donc la valeur de $\Pr\{F_r(j) \leq \tau\}$ diminue. Cette courbe nous permet donc de dimensionner τ en fonction de la probabilité $f = 0.005$ de défaillance d'une l'entité.

Nous utiliserons $r = 0.03$ et $\tau = 3$ dans les résultats suivants.

4.2.2 Comparaison des algorithmes

Nous avons présenté dans la section précédente deux conditions pour déterminer l'appartenance à M_k : l'une nécessaire, fournie par le corollaire 2 et implémentée par l'algorithme 3 et l'autre nécessaire et suffisante, fournie par le théorème 4 implémentée par l'algorithme 5. La condition du théorème 4 nécessite en plus la construction de

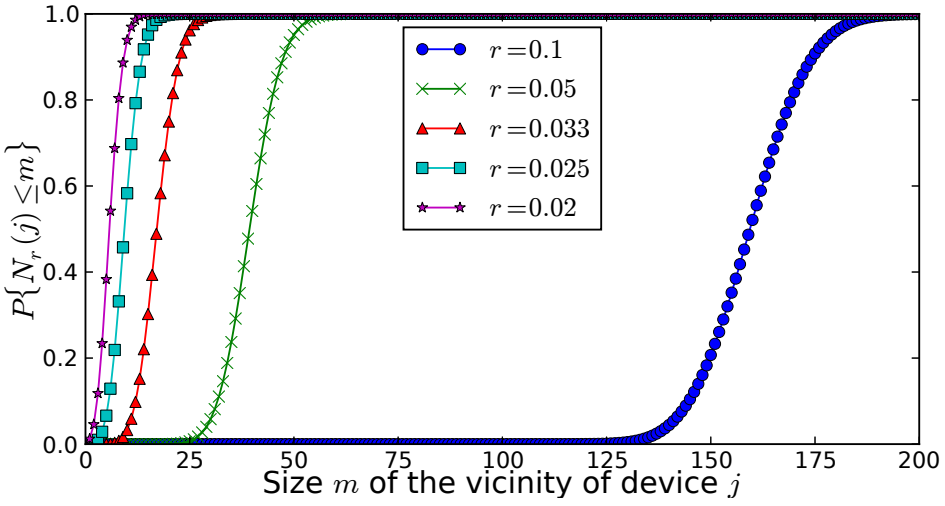


FIGURE 4.10 – $\Pr\{N_r(j) \leq m\}$ en fonction de la taille m du voisinage d'une entité dans l'espace des qualités pour différentes valeurs de rayon de cohérence r. On a $n = 1000$.

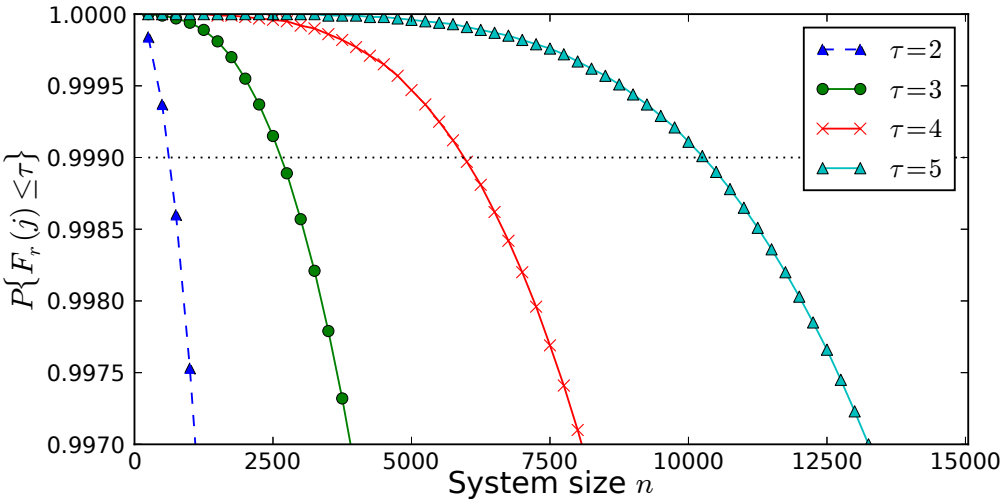


FIGURE 4.11 – $\Pr\{F_r(j) \leq \tau\}$ en fonction du nombre n d'entités supervisées pour différents seuils de densités τ. Le rayon de cohérence r valant 0.03 et la probabilité de faute isolée f de 0.005.

TABLE 4.1 – Répartition moyenne des entités de A_k dans I_k, M_k et U_k pour $G = 0$, $A = 20$, $n = 1000$, $r = 0.03$, $\tau = 3$ et une taille moyenne de A_k de 95.7.

	I_k	M_k	U_k
Algorithme 5 (Théorème 4)	2.54%	88.74%	8.72%
Algorithme 3 (Corollaire 2)	2.54%	88.34%	9.12%

toutes les collections d'ensembles contenant les entités de $L_k(j)$ ayant des mouvements τ-dense.

Il est donc intéressant de comparer ces deux approches en termes de complexité de calcul et de couverture des cas. Le théorème 4 fournit une condition nécessaire et suffisante pour caractériser l'appartenance d'une entité $j \in A_k$ à l'ensemble M_k. On compare les résultats de l'algorithme 3 implémentant le corollaire 2 par rapport aux résultats fournis par l'algorithme 5 implémentant le théorème 4.

Des vérités synthétiques comprenant le plus possible d'anomalies massives sont générées. Pour cela, on fixe le paramètre $G = 0$. Quelques anomalies isolées apparaissent dans les simulations dues à la faible densité du voisinage de certaines entités. Pour $n = 1000, r = 0.03$ et $\tau = 3$, ce nombre est négligeable.

Le tableau 4.1 illustre les résultats de caractérisation des approches pour des simulations avec $A = 20$ anomalies générées impactant ainsi $|A_k| = 96$ entités. On constate que conformément à ce qui est attendu, peu d'entités (moins de 3%) subissent des fautes isolées. De plus, 88% des fautes sont effectivement capturées par le corollaire 2 tandis que le théorème 4 n'en capture que 0.4% de plus. Cela signifie donc que le corollaire 2 couvre la quasi totalité des cas pour lesquels on peut caractériser de manière certaine l'appartenance à M_k et donc le taux de mauvaise caractérisation lors de l'utilisation du théorème 3 est négligeable. Dans le cas d'une mauvaise caractérisation commise par l'algorithme 3, l'entité concernée est alors attribuée à U_k. Comme nous l'avons vu auparavant, la figure 3.6 illustre le type de cas de mauvaise détection effectuée par le corollaire 2.

On constate de plus qu'un faible pourcentage (8%) des entités de A_k est en état indéterminé. Ce pourcentage est du à la proximité d'anomalies massives dans la vérité synthétique générée.

Exemple 15 *La figure 4.12 illustre ce cas. Deux anomalies massives* $C_1 = \{1, 3, 4, 6\}$ *(en blanc) et* $C_2 = \{2, 5, 7, 8\}$ *(en noir) avec* $\tau = 3$*. Celles-ci sont suffisamment proches l'une de l'autre pour qu'on ait*

$$\forall \ell \in \{2, 4, 6, 7\}, \overline{\mathcal{W}}_k(\ell) = \{C_1, C_2\}.$$

Le corollaire 2 s'applique pour les entités $2, 4, 6$ *et* 7*. En effet, si on considère l'entité* 2 *par exemple, on a* $J_k(2) = \{2, 4, 6, 7\}$*. On a* $|J_k(2)| = 4 > \tau$ *donc l'ensemble* $\{2, 4, 6, 7\}$ *a un mouvement* τ*-dense. De plus, on a* $\{2, 4, 6, 7\} \subseteq J_k(2)$ *donc le corollaire 2 s'applique et on a* $2 \in M_k$*. De la même manière on montre que* $\{2, 4, 6, 7\} \subseteq M_k$*.*

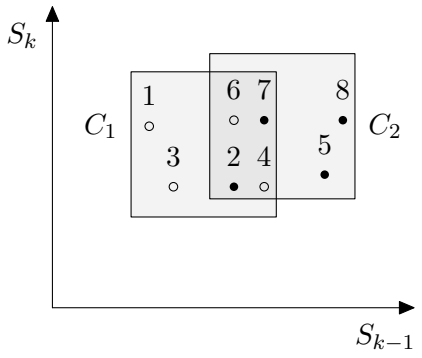


FIGURE 4.12 – Superposition d'anomalies massives menant à des états indéterminés.

À l'inverse, si on considère les entités $1, 3, 5$ *et* 8*, on a* $J_k(1) = J_k(3) = \{1, 3\}$ *et* $J_k(5) = J_k(8) = \{5, 8\}$*. On a* $|J_k(1)| = |J_k(5)| = 2 \leq \tau$*. Il n'existe donc pas de sous-ensemble de* $J_k(1)$ *ou de* $J_k(5)$ *ayant un mouvement* τ*-dense : le corollaire 2 ne s'applique donc pas.*

En revanche, si on considère par exemple l'entité 1 *et la famille* $\mathcal{F} = \{\{2, 4, 5, 6, 7, 8\}\}$ *le corollaire 3 s'applique et nous avons alors* $1 \in U_k$*. De la même manière, on montre qu'on a* $U_k = \{1, 3, 5, 8\}$*.*

Les états indéterminés sont donc dus à la proximité d'anomalies générées, c'est-à-dire à la probabilité de superposer des anomalies générées. Cette probabilité est liée au nombre n d'entités dans E ainsi qu'au nombre d'anomalies générées dans l'intervalle de temps $[k-1, k]$.

Le tableau 4.2 montre le coût en termes de calcul des algorithmes. Dans ce cas, nous avons généré $A = 20$ anomalies dans la simulation. En moyenne, 95.7 entités ont perçu une défaillance.

Les deux solutions nécessitent de calculer pour chaque entité $j \in A_k$ la famille $\mathcal{M}(j)$ contenant les ensembles r-cohérents impliquant j. Les deux approches nécessitent ensuite de tester les éléments de $\mathcal{M}(j)$ afin de tester le théorème 2. La première colonne fournit le nombre moyen d'ensembles de $\mathcal{M}(j)$: les entités impactées par une faute isolée appartiennent en moyenne à $1,85$ ensemble ayant des mouvements r-cohérents maximaux dans ces simulations. Ceci est cohérent avec le paramètre $G = 0$ car les seules anomalies isolées générées sont dues au faible voisinage de certaines entités.

Dans le cas d'une faute non isolée, les deux approches construisent alors $J_k(j)$ et $L_k(j)$ afin de tester le corollaire 2. La seconde colonne du tableau décrit le nombre moyen d'ensembles de $\overline{\mathcal{W}}_k(j)$ permettant de déduire le type de faute dont la défaillance est perçue par les entités supervisées. Ces deux premières colonnes représentent le coût moyen du corollaire 2 pour les paramètres de simulation utilisés.

À l'inverse, lors de l'utilisation du théorème 4, celui-ci nécessite de tester potentiellement toutes les collections possibles comprenant les éléments de $L_k(j)$ afin de déterminer si l'entité j appartient à U_k ou M_k. Le nombre moyen de collection testé

TABLE 4.2 – Nombre moyen d'ensembles testés de calcul pour chaque entité de I_k, M_k, U_k pour $G = 0$, $A = 20$, $n = 1000$ $r = 0.03$, $\tau = 3$ et $|A_k| = 95.7$.

	I_k	M_k	U_k
Algorithme 5 (Théorème 4)	1.85	2 450 150	31 108
Algorithme 3 (Corollaire 2)	1.85	1.17	5.57

avant cette décision est fourni dans les colonnes 3 et 4 du tableau. Cette approche s'avère donc extrêmement coûteuse comparée au coût d'utilisation du corollaire 2. De plus, son utilisation n'apporte qu'un gain de 0.4%. Dans la suite, nous utiliserons donc l'algorithme 3.

4.2.3 Impact de la fréquence d'échantillonnage des états du système

Notre modélisation nous permet de détecter aussi rapidement que possible le type de faute dont la défaillance est perçue par les entités supervisées. Nous étudions à présent le nombre d'entités en état indéterminé en fonction de la fréquence d'échantillonnage des états du système. Comme nous venons de le voir, les états indéterminés se produisent lorsque plusieurs fautes impactent des entités proches dans l'espace des qualités.

Pour un système comportant $n = 1000$ entités supervisées avec une probabilité de faute isolée $f = 0.005$, la figure 4.13 illustre l'évolution de $|U_k|/|A_k|$ en fonction du nombre de fautes A générées dans le système pour différentes valeurs de G. Cette proportion désigne la proportion d'entités supervisées en état indéterminé par rapport au nombre d'entités percevant une défaillance. Cette figure valide le fait que plus le nombre de fautes impactant le système dans l'intervalle de temps $[k-1, k]$ augmente, plus le nombre d'état indéterminés augmente.

Ainsi, si les entités sont capables d'échantillonner les états du système de manière suffisamment fréquente pour qu'il n'y ait au plus qu'une seule faute impactant le système dans l'intervalle de temps $[k-1, k]$, la superposition de fautes est impossible et donc le nombre d'états indéterminés devient nul. De plus, si la faute impactant le système est une faute massive alors toutes les entités j impactées par cette faute calculent le même ensemble $J_k(j)$ et on a alors $L_k(j) = \emptyset$. Dans ce cas, le théorème 3 décide efficacement pour toutes les entités impactées par cette faute.

La lecture de la figure 4.13 montre également le faible impact de la distribution du type de fautes sur l'état du système. En effet, un système ne subissant que des fautes massives engendre plus d'états indéterminés qu'un système ne subissant que des fautes isolées. Lorsque que $G = 0$, quasiment toutes les anomalies générées sont des anomalies massives augmentant ainsi la probabilité que celles-ci se superposent. À l'inverse, lorsque $G = 1$, la majorité des anomalies générées sont des anomalies isolées. Lorsque plus de τ entités appartenant à des anomalies isolées sont suffisamment proches dans l'espace des qualités E, ces anomalies sont fusionnées en une anomalie massive afin de respecter la restriction **R3** du modèle. Ceci donne lieu à l'apparition

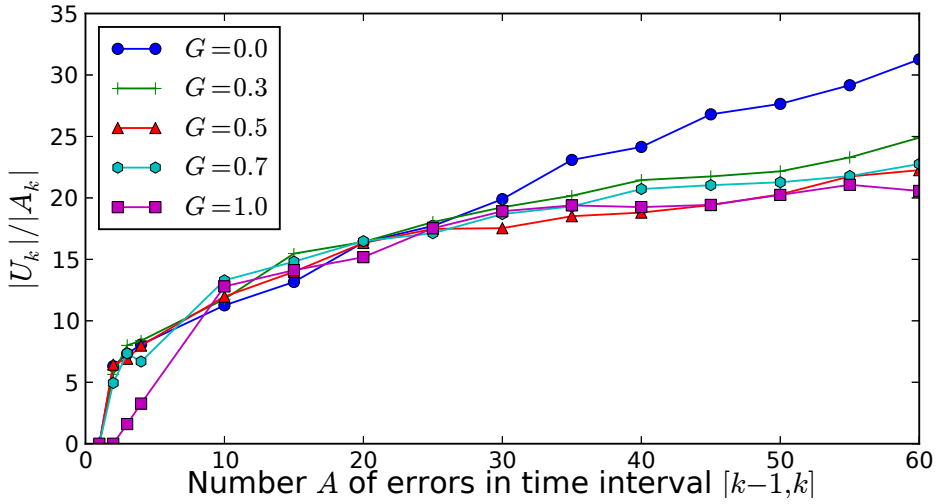


FIGURE 4.13 – Proportion d'entités en état indéterminé $|U_k|/|A_k|$ en fonction du nombre A de fautes dans le système dans l'intervalle de temps $[k-1, k]$ et de la proportion de fautes massives G. Le système est composé de $n = 1000$ entités et la probabilité de faute isolée est $f = 0.005$.

d'anomalies massives dans la vérité synthétique générée, permettant ainsi l'apparition d'états indéterminés.

4.2.4 Pertinence du modèle

Comme nous l'avons évoqué dans la section 4.2.1, il arrive qu'on soit amené à modifier la vérité synthétique générée pour la simulation afin qu'elle respecte la restriction **R3** du modèle. Celle-ci impose que dans un ensemble ayant un mouvement τ-dense, il y ait au minimum une entité ayant subi une faute massive. Cela signifie donc qu'un ensemble ayant un mouvement τ-dense ne peut pas être constitué uniquement d'entités impactées par une faute isolée.

Afin de mesurer l'écart de ce modèle à la vérité, de nouvelles vérités synthétiques sont générées et non modifiées même si celles-ci ne respectent pas la condition **C2** de la définition 8 de partition d'anomalies. On mesure alors les taux de mauvaises détections du type de fautes dans ce cas, c'est-à-dire la proportion des entités ayant subi une faute isolée dans la vérité synthétique et pour lesquelles l'algorithme 3 rend "Massive".

Par construction de la caractérisation, il est impossible pour une entité ayant subi une faute massive que l'algorithme conclut à une faute isolée. En effet, si la vérité synthétique comporte des entités appartenant à une anomalie massive C, il existe au minimum un ensemble ayant un mouvement τ-dense contenant ces entités : il s'agit de l'ensemble C. Par conséquent le théorème 2 ne s'applique pas. Il est donc impossible pour une entité ayant subi une faute massive que l'algorithme conclue à une faute isolée.

La figure 4.14 illustre cette proportion en fonction du nombre d'anomalies générées dans le système pour différentes proportions d'anomalies isolées/massives. On constate sur cette figure que pour toutes les proportions d'anomalies isolées/massives, le nombre de mauvaises détections croît jusqu'à 10 puis se stabilise.

De plus, on s'aperçoit que lorsque seules des anomalies massives ($G = 0$) sont générées, la proportion de mauvaises détections est très faible. Celles-ci sont dues à des anomalies isolées générées faute d'un voisinage suffisamment dense, puis à la superposition d'une seconde anomalie massive. Cette situation est illustrée par la figure 4.12.

À l'inverse, lorsque seules des anomalies isolées ($G = 1$) sont générées, la proportion de mauvaises détections augmente jusqu'à environ 10%. Dans ce cas, seules les entités supervisées subissent des fautes tandis que le reste du système est parfait et ne subit aucune faute. Ainsi, dans le pire des cas, ce modèle ne s'éloigne que de 10% de la réalité pour les entités ayant subi une faute isolée.

La figure 4.15 illustre l'évolution de la proportion d'entités en état indéterminé en fonction du nombre de fautes impactant le système. Celle-ci reste totalement stable, traduisant le fait que la restriction **R3** du modèle n'a aucun impact sur la qualité de la décision. En effet, lors de l'application du corollaire 3, l'existence d'une collection d'ensemble dans laquelle l'entité considérée appartient à un ensemble au mouvement non τ-dense suffit à amener à un état indéterminé. Cette collection n'est donc basée que sur les positions des entités proches aux instants $k-1$ et k et non pas sur la manière dont ces entités se sont déplacées. Il est donc normal que la restriction **R3** du modèle n'ait aucun impact sur la proportion d'états indéterminés.

Ces résultats montrent ainsi que les contraintes imposées pour ce modèle ne sont pas trop restrictives. Ces contraintes nous ont permis de construire un formalisme nous permettant d'établir une caractérisation des fautes suivant leur impact. Cependant, les simulations nous montrent qu'en l'absence de ces contraintes, le modèle s'applique dans une très grande majorité des cas et n'engendre qu'une faible proportion de mauvaises détections.

4.3 Discussion

L'algorithme DBSCAN [EKSX96] dont le pseudo-code est décrit dans l'algorithme 7 est un algorithme de partitionnement de données basé sur la densité des données dans l'espace. L'algorithme partitionne un ensemble de points S en différents clusters en fonction d'une mesure de distance D. Tout comme notre approche, cet algorithme requiert deux paramètres ε et m. L' ε-voisinage $N_\varepsilon(p)$ d'un point $p \in S$ est défini par :

$$N_\varepsilon(p) = \{q \in S \mid D(p,q) \leq \varepsilon\}.$$

Un point $p \in S$ est directement atteignable depuis un autre point $q \in S$ si les deux conditions suivantes sont vérifiées :

$$\begin{array}{rl} p \in N_\varepsilon(q) & \textit{i.e.}, p \text{ appartient à } \varepsilon\text{-voisinage de } q, \\ |N_\varepsilon(q)| \geq m & \textit{i.e.}, \text{l'}\varepsilon\text{-voisinage de } q \text{ est suffisamment peuplé.} \end{array}$$

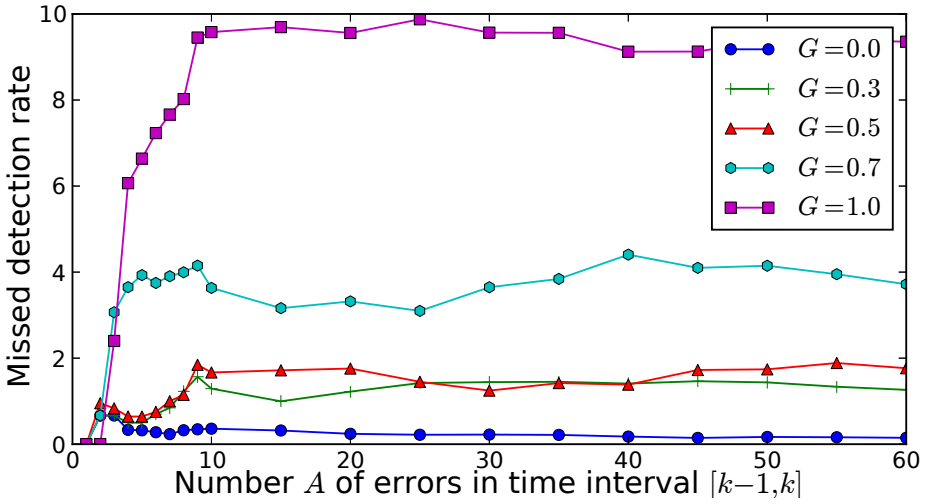


FIGURE 4.14 – Proportion de mauvaises détections en fonction du nombre A de fautes générées dans le système sans restriction **R3**. Le système est composé de $n = 1000$ entités. La probabilité de faute isolée vaut $f = 0.005$.

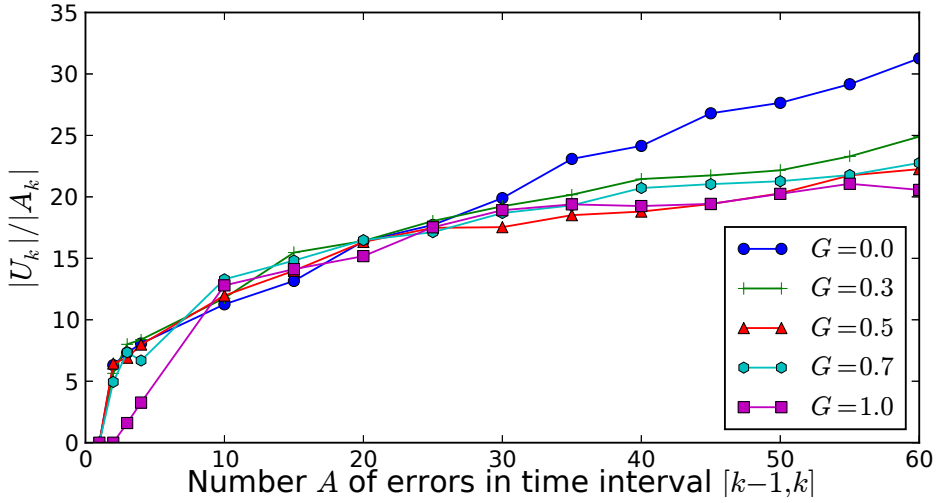


FIGURE 4.15 – Proportion d'entités en état indéterminé $|U_k|/|A_k|$ en fonction du nombre A de fautes dans le système sans restriction **R3** dans l'intervalle de temps $[k-1, k]$ et de la proportion de fautes massives G. Le système est composé de $n = 1000$ entités et la probabilité de faute isolée vaut $f = 0.005$.

Un point $p \in S$ est atteignable depuis un autre point $q \in S$ s'il existe un chemin $(p_1, \ldots, p_n)$ telle que :

$$p_1 = q,$$
$$p_n = p,$$
$$\forall 1 \leq i < n, p_{i+1} \text{ est directement atteignable depuis } p_i.$$

Deux points $p, q \in S$ sont connectés s'il existe $o \in S$ tel que p et q soient atteignables depuis o. Un ensemble $C \subseteq S$, $C \neq \emptyset$ est un cluster s'il vérifie les deux conditions suivantes :

Maximalité : $\forall p, q \in S, p \in C$, q est atteignable depuis $p \Longrightarrow q \in C$

Connectivité : $\forall p, q \in C$, p et q sont connectés.

Enfin, la notion de cluster permet d'introduire la notion de bruit. Le bruit représente une donnée erronée ou aberrante vis-à-vis du reste des données. Étant donnés $C_1, \ldots, C_\ell$ les clusters de S, l'ensemble B du bruit est défini par :

$$B = \{\ell \in S \mid \forall C_i, \ell \notin C_i\}.$$

On note immédiatement la similarité existante entre la notion de bruit de DBSCAN et l'ensemble I_k de notre approche. En effet, les entités que DBSCAN classifie comme bruit sont celles pour lesquelles le théorème 2 s'applique.

Les approches diffèrent ensuite sur la gestion des autres entités. DBSCAN cherche à regrouper les entités au sein de clusters distincts tandis que dans notre approche on souhaite caractériser l'appartenance d'une entité à un ensemble ayant un mouvement τ-dense, non τ-dense ou en état indéterminé.

De plus, le partitionnement engendré par DBSCAN n'est pas déterministe.

Exemple 16 *La figure 4.16 illustre deux exécutions de l'algorithme menant à des partitionnements différents. On considère l'ensemble d'entités* $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$ *de la figure 4.16(a). La figure 4.16(b) représente les* ε*-voisinages de chaque entité de* S *où la présence d'une arrête* (i, j) *signifie que la distance de* i *à* j *est inférieure à* ε*. Seules les entités* 3 *et* 5 *représentées en noir sont centrales. On a* $N_\varepsilon(3) = \{1, 2, 3, 4\}$ *et donc* $|N_\varepsilon(3)| \geq m = 4$*. De même, on a* $N_\varepsilon(5) = \{4, 5, 6, 7, 8\}$ *et donc* $|N_\varepsilon(5)| \geq m = 4$*. À l'inverse,* $N_\varepsilon(4) = \{3, 4, 5\}$*, donc* $|N_\varepsilon(4)| < m$*, les entités* 3 *et* 5 *ne sont donc pas connectées et n'appartiennent pas au même cluster. En revanche, l'entité* 4 *est directement atteignable depuis* 3 *et* 5*. Cette situation conduit à deux partitionnements possibles. Une première exécution de l'algorithme DBSCAN construit les deux clusters* C_1, C_2 *de la figure 4.16(c), alors qu'une autre exécution possible construit les clusters* C'_1, C'_2 *de la figure 4.16(d).*

À l'inverse, dans notre approche, la caractérisation effectuée est valable pour tous les partitionnements possibles vérifiant la définition 8.

Exemple 17 *Étant donnée la configuration décrite dans la figure 4.16, l'algorithme 3*

Algorithme 7 : Algorithme DBSCAN

```
1  Algorithme dbscan()
2  |  F ← ∅;
3  |  pour chaque p non visité de S faire
4  |  |  marquer p comme visité;
5  |  |  N_ε(p) ← epsilon-Voisinage(S,p, ε);
6  |  |  si |N_ε(q)| < m alors
7  |  |  |  marquer p comme Bruit;
8  |  |  sinon
9  |  |  |  C ← ∅;
10 |  |  |  expandCluster(S, F, p, N_ε(p), C, ε, m);
11 |  |  |  F ← {C};
12 |  |  fin
13 |  fin
1  Procédure expandCluster(S, F, p, N_ε(p), C, ε, m)
2  |  C ← C ∪ p;
3  |  pour chaque q ∈ N_ε(p) faire
4  |  |  si q n'a pas été visité alors
5  |  |  |  marquer q comme visité;
6  |  |  |  N_ε(q) ← epsilon-Voisinage(S,q, ε);
7  |  |  |  si |N_ε(q)| ≥ m alors
8  |  |  |  |  N_ε(p) ← N_ε(p) ∪ N_ε(q);
9  |  |  |  fin
10 |  |  fin
11 |  |  si ∀C' ∈ F, q ∉ C' alors
12 |  |  |  C ← C ∪ q;
13 |  |  fin
14 |  fin
1  Fonction epsilon-Voisinage(S, p, ε)
2  |  retourner {q ∈ S | ‖q − p‖ ≤ ε};
```

construit les ensembles suivants :

$$\begin{aligned}
D_k(1) = \{1,2,3,4\}, J_k(1) = \{1,2\} \\
D_k(2) = \{1,2,3,4\}, J_k(2) = \{1,2\} \\
D_k(3) = \{1,2,3,4,5,6\}, J_k(3) = \{1,2,3\} \\
D_k(4) = \{1,2,3,4,5,6,7,8\}, J_k(4) = \{1,2,3,4\} \\
D_k(5) = \{3,4,5,6,7,8\}, J_k(5) = \{5,6,7,8\} \\
D_k(6) = \{3,4,5,6,7,8\}, J_k(6) = \{5,6,7,8\} \\
D_k(7) = \{4,5,6,7,8\}, J_k(7) = \{7,8\} \\
D_k(8) = \{4,5,6,7,8\}, J_k(8) = \{7,8\}
\end{aligned}$$

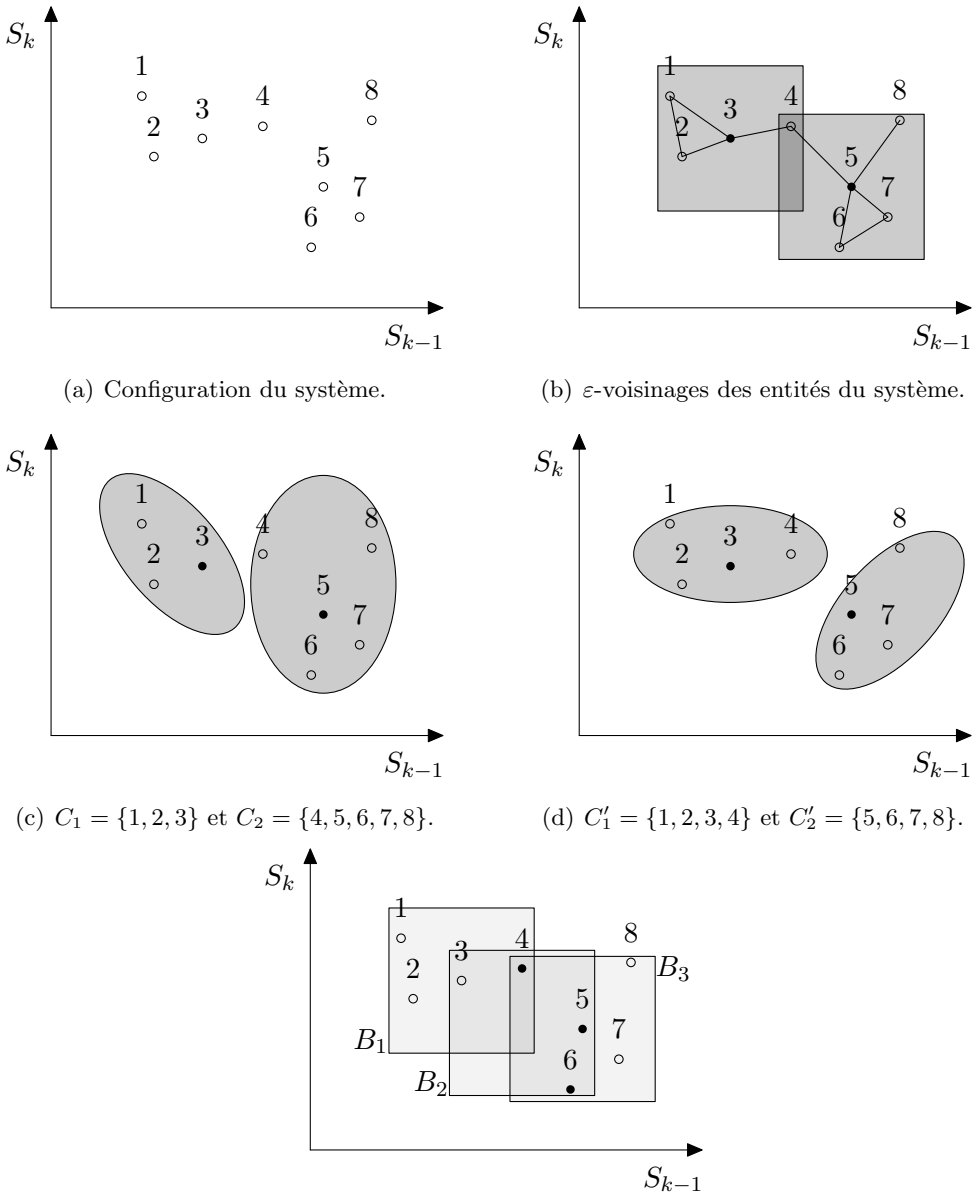
Par application du corollaire 2, on a :

$$I_k = \emptyset, M_k = \{4,5,6\} \text{ et } U_k = \{1,2,7,8\}.$$

Le calcul des clusters dans DBSCAN repose sur la construction de l'ε-voisinage $N_\varepsilon(p)$ de chaque entité $p \in S$. Cet ensemble est centré sur la position de p dans E. À l'inverse, notre approche repose sur l'existence d'ensembles r-cohérents. Comme l'indique la propriété 1, cette notion n'est pas centrée sur une position en particulier et offre ainsi une souplesse dans le calcul et est ainsi moins sensible à la position effective de chacune des entités. Dans la figure 4.16(a), la distance séparant les entités 3 et 5 est inférieure à $2r$, permettant ainsi à notre approche de construire l'ensemble $B_2 = \{3,4,5,6\}$ (voir figure 4.16(e)), ce qui est impossible dans le cas de DBSCAN car il n'y a pas d'entité présente dans l'intersection des boules de rayon r centrées sur les positions de $3,4,5,6$. L'algorithme DBSCAN se montre donc plus contraint dans sa recherche de clusters que ne l'est notre approche.

D'autre part, DBSCAN distingue deux types d'entités au sein d'un cluster : les entités centrales pour lesquelles l'ε-voisinage contient plus de m entités et les entités du bord du cluster qui sont atteignables depuis une entité centrale. Un cluster DBSCAN est constitué d'entités connectées par un chemin dense, c'est-à-dire une succession d'entités centrales. Ainsi deux entités peuvent appartenir à un même cluster alors que la distance qui les sépare peut être arbitrairement grande. À l'inverse, dans l'approche décrite dans ce chapitre, la caractérisation repose uniquement sur le voisinage des entités à $4r$. Ainsi, alors que DBSCAN a besoin d'une vision totale de la configuration du système, notre approche ne s'appuie que sur cette vision locale. Cela signifie donc qu'il n'est pas nécessaire que les entités effectuent leurs mesures de manière totalement synchronisée. Il est en effet suffisant pour une entité percevant une défaillance d'être capable de déterminer les éventuelles autres entités ayant perçu cette défaillance. À l'inverse, dans le cas de DBSCAN, une synchronisation forte est requise afin de construire l'état du système à l'instant k dans son intégralité.

L'une des limitations majeures de l'algorithme DBSCAN est son incapacité à capturer efficacement des clusters de densités différentes. En effet, DBSCAN nécessite une vision globale de la configuration du système et par conséquent les paramètres ε et m



(a) Configuration du système.

(b) ε-voisinages des entités du système.

(c) $C_1 = \{1, 2, 3\}$ et $C_2 = \{4, 5, 6, 7, 8\}$.

(d) $C'_1 = \{1, 2, 3, 4\}$ et $C'_2 = \{5, 6, 7, 8\}$.

(e)

FIGURE 4.16 – Différentes exécutions de l'algorithme DBSCAN pour $m = 4$.

sont les mêmes pour toutes les entités du système. À l'inverse, notre approche ne reposant que sur la connaissance du voisinage des entités percevant une défaillance, celle-ci peut tout à fait gérer des paramètres r et τ différents suivant les régions de l'espace des qualités. De cette manière, notre approche peut caractériser les fautes de manière différente suivant la localisation des entités dans l'espace des qualités et ainsi s'adapter à des densités de population différentes.

Bien que de prime abord, notre approche semble similaire à DBSCAN, celles-ci diffèrent largement dans leurs finalités. En effet, notre approche cherche à caractériser l'appartenance à I_k, U_k ou M_k pour tous les partitionnements possibles respectant la définition 8 tandis que DBSCAN s'attache à exhiber un partitionnement des entités en fonction des paramètres ε et m. De plus, DBSCAN repose sur une approche centralisée nécessitant la connaissance de la totalité du système tandis que notre approche est décentralisable par nature. Enfin, notre approche offre beaucoup de souplesse que DBSCAN dans la mesure où les paramètres r et τ peuvent être localement réglés, contrairement à DBSCAN où les paramètres ε et m sont globaux.

4.4 Conclusion

Nous avons présenté dans le chapitre précédent un modèle et des conditions permettant la résolution du problème 2. Ce modèle s'appuie uniquement sur la connaissance du voisinage dans l'espace des qualités des entités percevant des défaillances. Par conséquent, le caractère local des informations requises pour la caractérisation rend cette approche décentralisable.

Dans ce chapitre, nous avons présenté les différents algorithmes nécessaires à la mise en œuvre des conditions décrites dans le chapitre 3 de manière décentralisée. Les différents algorithmes ont été comparés et évalués au travers de simulation.

Les conditions permettant la résolution du problème 2 ne s'appuient que sur la connaissance du voisinage dans l'espace des qualités des entités percevant des défaillances. Le chapitre suivant décrit une architecture auto-organisante visant à fournir à chaque entité supervisée l'accès à cette connaissance requise. De plus, il décrit la complexité des différentes opérations nécessaires au fonctionnement de cette architecture ainsi que son utilisation pour l'implémentation des algorithmes décrits dans ce chapitre.

Bibliographie

[ABL+14] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola : Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, juin 2014.

[EKSX96] M. Ester, H. P. Kriegel, J. Sander et X. Xu : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Dans *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 226–231, 1996.

Chapitre 5

FixMe : Une architecture auto-organisante pour la caractérisation de fautes

Les fautes peuvent apparaître dans le système et leur impact peut être perçu par différentes entités. On cherche alors à distinguer les fautes qui impactent un petit nombre d'entités (inférieur à un paramètre fixé τ) de celles à plus large impact (*i.e.*, qui impactent strictement plus que τ entités supervisés). Dans le premier cas, on parlera alors de *faute isolée* tandis que dans le second, on parlera de *faute massive*.

Cette distinction repose sur la forte corrélation présente entre les mesures effectuées par les entités. En effet, des entités mesurant des valeurs de performances similaires peuvent être susceptibles de percevoir les mêmes défaillances. On traduit cette intuition par l'hypothèse suivante : si les mesures effectuées avant la perception de la défaillance sont proches et qu'elles le sont encore après la perception de celle-ci, alors la défaillance est due à la même faute. Cette hypothèse est modélisée par une distance seuil entre les positions des entités dans l'espace des qualités : s'il existe une boule de rayon r contenant les positions de ces entités dans l'espace des qualités aux instants $k-1$ et k alors ces entités ont perçu la même défaillance.

Nous avons vu dans le chapitre 3 qu'il était impossible de déterminer avec exactitude pour chaque entité percevant une défaillance si celle-ci était due à une faute isolée ou à une faute massive. Néanmoins, il est en revanche possible de distinguer les fautes isolées, massives et les états indéterminés. Cette distinction repose sur la possibilité de construire pour chaque entité $j \in [\![1, n]\!]$ percevant une défaillance la famille $\overline{\mathcal{W}}_k(j)$ des ensembles ayant un mouvement τ-dense maximal dans l'intervalle de temps $[k-1, k]$. Ainsi, lorsqu'une entité $j \in [\![1, n]\!]$ perçoit une défaillance, il est nécessaire de déterminer l'ensemble des entités percevant une défaillance similaire, c'est-à-dire dont les variations de mesures de qualité sont fortement corrélés à celles mesurées par j. Cet ensemble noté $N(j)$ est défini comme suit :

$$N(j) = \{\ell \in [\![1, n]\!] \setminus \{j\} \mid \|p_{k-1}(j) - p_{k-1}(\ell)\| \leq 2r, \|p_k(j) - p_k(\ell)\| \leq 2r\}.$$

Ce chapitre décrit FixMe une architecture visant à fournir à chaque entité supervisée j ayant perçu une défaillance, l'ensemble des entités susceptibles d'avoir perçu la même défaillance et ainsi mettre en œuvre la caractérisation proposée aux chapitres précédents.

5.1 Problématique

On cherche à fournir une architecture permettant à chaque entité j supervisée de trouver efficacement les entités qui appartenaient au voisinage dans l'espace des qualités E de j à l'instant $k-1$ et qui sont encore dans le voisinage de j à l'instant k.

Le caractère local de ces informations nous oriente vers une gestion totalement décentralisée et auto-organisée de ces entités. De cette manière chaque entité peut alors conserver un minimum d'information locale tout en étant capable de trouver les informations nécessaires à la mise en œuvre de la caractérisation de faute.

On cherche donc à mettre en place une structure logique (*overlay*) permettant d'interconnecter les entités supervisées percevant des qualités similaires. Ainsi, à chaque variation de qualité, ces entités se déplacent dans l'espace des qualités et leur voisinage change. Il semble donc plus adapté d'effectuer une recherche sur ce voisinage lors de la perception d'une défaillance plutôt que de maintenir l'information du voisinage auprès de chaque entité supervisée. Cette architecture doit donc être résistante au dynamisme des entités.

Nous considérons un réseau large échelle et nous faisons l'hypothèse qu'il n'y a donc pas de faute susceptible d'impacter l'ensemble des entités du système. Par conséquent, les entités ne percevant pas de défaillances vont assurer la connectivité de la structure logique utilisée tandis que celles percevant une défaillance se replaceront dans celle-ci afin de déterminer les informations nécessaires à la caractérisation. On souhaite gérer un très grand nombre d'entités et déterminer au plus vite le type de faute impactant les entités supervisées. La recherche des voisinages aux différents instants doit donc être efficace. Cette dernière contrainte nous invite à laisser de côté les réseaux non structurés pour s'orienter vers une approche structurée du type table de hachage distribuée (DHT).

Les systèmes qui utilisent des approches de ce type organisent les entités en fonction de leurs identifiants issus d'une fonction de hachage $\mathfrak{h}$ [RFH+01, SMK+01]. Les fonctions de hachage sont des fonctions qui projettent les données fournies en entrée dans un espace de taille réduite. Par exemple, la fonction SHA-1 [EJ01] projette une donnée quelconque dans $[\![0,1]\!]^{160}$. Ces fonctions ont la propriété de distribuer uniformément les identifiants sur l'espace d'adressage. Ainsi, le placement des entités dans les systèmes utilisant ce type d'approche est également uniforme. L'efficacité de ces approches repose sur l'uniformité de ce placement. La figure 5.1 illustre le comportement de CAN en présence d'entités uniformément distribuées. On constate alors que les zones ont une taille sensiblement équivalente. La performance de CAN repose sur ce découpage en zones, comme nous l'avons vu dans la section 2.1.1. Toute DHT fournit une opération `lookup`. Cette opération permet de trouver, à partir d'une entité source, une donnée destination, recherchée dans l'espace géré par la DHT. Dans le cas de CAN, cette

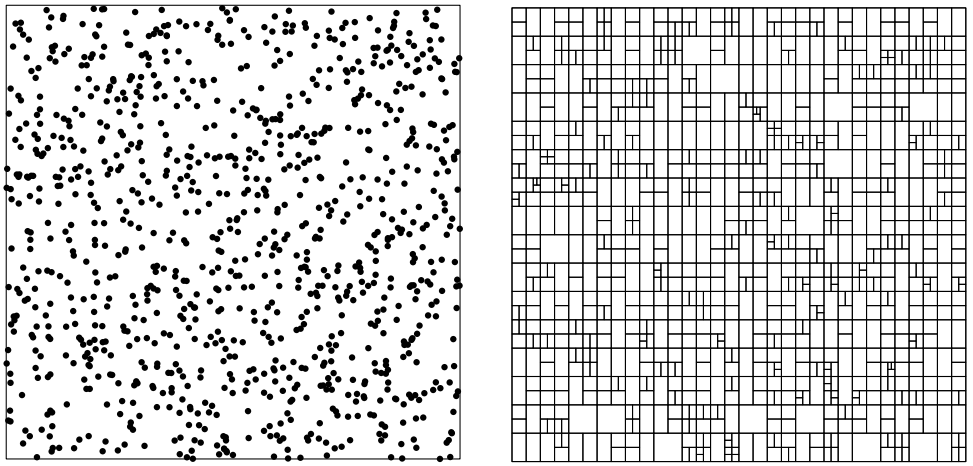


FIGURE 5.1 – Distribution uniforme des nœuds (à gauche) et topologie CAN induite (à droite)

opération consiste à traverser les zones de proche en proche de la source à la destination. Lorsque n entités sont uniformément réparties dans l'espace de dimension d géré par la DHT, une opération de `lookup` nécessite de traverser $\mathcal{O}\left(n^{1/d}\right)$ zones.

Cependant, le comportement de ces approches se dégrade si les entités ne sont plus uniformément distribuées sur l'espace d'adressage. Par exemple, si on place les entités dans CAN [RFH+01] suivant une distribution différente, le comportement de celui-ci se dégrade. Ainsi la figure 5.2 illustre le comportement de CAN face à une distribution des entités non uniforme. Dans ce cas, une disparité dans la répartition des tailles des cellules apparaît et le comportement de CAN se dégrade. En effet, le découpage des zones n'étant plus régulier, cela augmente significativement le nombre de zones traversées lors d'une recherche et engendre donc une chute d'efficacité.

Le recours aux fonctions de hachage permet donc d'uniformiser les entités sur l'espace d'adressage et ainsi d'assurer l'efficacité de ces approches. Cependant, cette efficacité est obtenue au prix de la logique applicative. En effet, il n'y a aucune relation entre une donnée et son image par la fonction de hachage. Ainsi, si la distance entre deux données ou entités est faible, on n'a aucune garantie sur la distance entre leurs images.

Dans notre cas, on cherche à fournir une architecture permettant à chaque entité j supervisée de trouver efficacement les entités qui appartenaient au voisinage dans l'espace des qualités E de j à l'instant $k-1$ et qui sont encore dans le voisinage de j à l'instant k. Par conséquent, on ne peut donc avoir recours à ce type d'approche.

La section suivante décrit FixMe, une architecture auto-organisante permettant une recherche efficace tout en conservant la logique applicative.

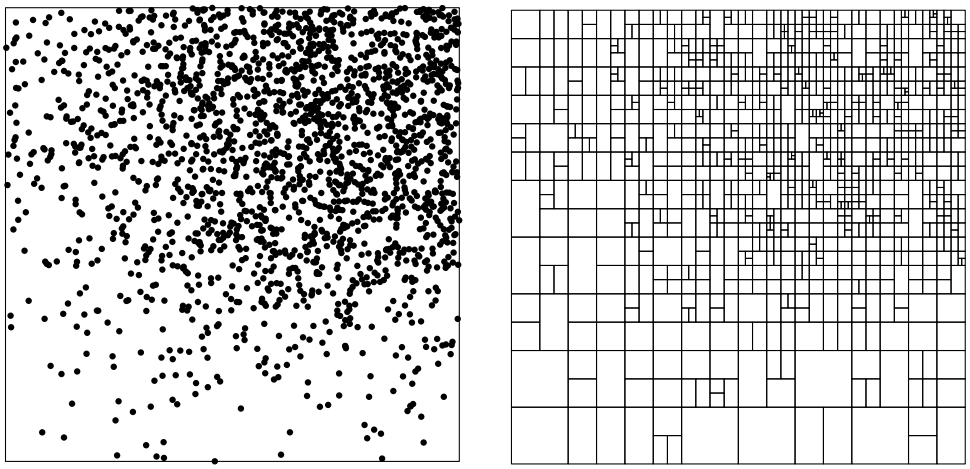


FIGURE 5.2 – Distribution non-uniforme des nœuds (à gauche) et topologie CAN induite (à droite)

5.2 Architecture de l'espace des qualités

5.2.1 Éléments de l'architecture FixMe

On souhaite que les entités s'organisent dans FixMe afin que les entités mesurant des valeurs de performances proches soient proches également dans la topologie de FixMe. Ainsi, ces entités se placent dans un espace cartésien E de dimension d à structure torique. Cet espace est partitionné en un ensemble de sous-espaces élémentaires appelés *buckets*. Un bucket correspond au produit cartésien de d intervalles de longueurs respectives $\rho_1, \ldots, \rho_d$. On note $b_i = 1/\rho_i$ le nombre d'intervalles élémentaires de la dimension i. Le tableau 5.1 rassemble les notations employées dans ce chapitre.

À l'instant k, une entité j dans FixMe appartient à l'unique bucket contenant la position $p_k(j)$. Un bucket contenant plus de γ entités à l'instant k est appelé *seed*. Les entités de FixMe partitionnent dynamiquement l'espace E en zones appelées *cellules*. Ce partitionnement est tel que chaque cellule contient au plus un seed. De plus, les buckets sont indivisibles, ils constituent ainsi la plus petite subdivision possible de E. De manière plus formelle, un cellule est définie comme suit :

Définition 12 (Cellule) *Une cellule est un hyper-rectangle de buckets parmi lesquels au plus un bucket est un seed. Une cellule est totalement caractérisée par un ensemble ordonné de 2^d buckets appelés* coins*. Les coins sont ordonnés suivant l'ordre lexicographique.*

Exemple 18 *La figure 5.3 illustre les différents éléments nécessaires à la construction de FixMe pour $d = 2$ et $\rho_1 = \rho_2 = 1/16$. Les buckets correspondent aux carrés élémentaires, les seeds aux carrés noirs et les cellules aux rectangles de couleur.*

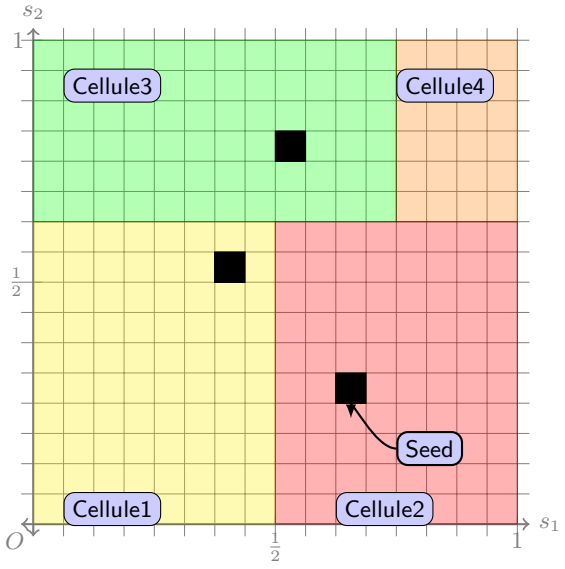


FIGURE 5.3 – Partitionnement de l'espace E de FixMe à $d = 2$ dimensions.

Conformément à la définition 12, les cellules 1*,* 2 *et* 3 *contiennent un seed tandis que la cellule* 4 *n'en contient pas.*

Toute DHT fournit une opération `lookup`. Cette opération décrit comment parcourir la topologie de la DHT afin de localiser efficacement une ressource dans le système. Ce parcours s'appuie sur l'utilisation d'une table de routage. Il s'agit d'une table contenant des couples (clé, destination). Pour chaque clé de la table de routage maintenue par une entité p, la destination associée contient l'entité (ou un ensemble d'entités) la plus proche de la clé, connue de p. Dans FixMe, cette opération est utilisée par une entité pour trouver le bucket auquel elle appartient. La connectivité et le routage dans FixMe sont exclusivement gérés par les entités appartenant à un seed.

Les entités présentes dans un seed maintiennent une table de routage assurant ainsi la connectivité de la topologie. Plus spécifiquement, cette table de routage associe à un bucket $B_{i,j}$ le seed responsable de la cellule contenant $B_{i,j}$.

Les entrées $B_{i,j}$ sont définies comme suit. Soit $(x_1, \ldots, x_d)$ les coordonnées du centre du seed, on considère la dimension i. Les entrées de la table de routage sont ordonnées dans l'ordre croissant suivant la i-ème coordonnée. On définit alors la clé $B_{i,j}$ comme la j-ème entrée $B_{i,j}^{(+)}$. On associe alors à cette clé le seed de la cellule contenant le point $(x_1, \ldots, x_{(i,j)}, \ldots, x_d)$ avec $x_{(i,j)} = \{x_i + 2^j \rho_d\}$. Pour $x \in \mathbb{R}$, la notation $\{x\}$ désigne ici la partie fractionnaire de x. De manière similaire on associe à $B_{i,j}^{(-)}$, la j-ème entrée $B_{i,j}$ dans le sens décroissant selon la dimension i, le seed de la cellule contenant le point $(x_1, \ldots, x_{(i,j)}, \ldots, x_d)$ avec $x_{(i,j)} = \{x_i - 2^j \rho_d\}$.

On définit alors $R_i \leq 1/2$ comme étant la distance séparant le bucket $B_{i,j}$ le plus lointain suivant la dimension i. Ce paramètre permet alors de faire un compromis entre

la taille de la table de routage et la répartition des entités dans E.

Proposition 1 *La table de routage sur E contient* $2\sum_{i=1}^{d}\lfloor\log_2(R_i/\rho_i)\rfloor$ *entrées.*

Preuve La distance entre le centre du seed et le bucket le plus loin accessible en un saut est borné par R_i. On définit alors $K_+^{(i)} = \max\{k \geq 1 \mid |x_i - x_{(i,+k)}| \leq R_i\}$ et $K_-^{(i)} = \max\{k \geq 1 \mid |x_i - x_{(i,-k)}| \leq R_i\}$. On a $|x_i - x_{(i,+k)}| = |x_i - x_{(i,-k)}| = 2^k\rho_i$ et donc $K_+^{(i)} = K_-^{(i)} = \lfloor\log_2(R_i/\rho_i)\rfloor$. On note $K^{(i)}$ cette valeur. Pour chaque dimension $i \leq d$, la table de routage contient donc $2K^{(i)}$ entrées. L'espace E est de dimension d, la table de routage maintenue par les entités du seed contient donc $\sum_{i=1}^{d} 2K^{(i)} = 2\sum_{i=1}^{d}\lfloor\log_2(R_i/\rho_i)\rfloor$ entrées. □

En plus de cette table de routage, les entités du seed maintiennent une table de prédécesseurs. Celle-ci correspond à une table de routage inversée contenant des couples (L, S). Un couple (L, S) est présent dans la table des prédécesseurs s'il existe un seed dans le système ayant une entrée dans sa table de routage $B_{i,j} = L$. Cette table des prédécesseurs n'est utilisée que dans les opérations `split` et `merge` et vise à notifier les prédécesseurs d'un changement topologique.

5.2.2 Opérations de FixMe

Nous décrivons à présent les cinq opérations nécessaires à la mise en œuvre de FixMe : `lookup`, `join`, `leave`, `split` et `merge`.

opération `lookup` Lorsqu'une entité souhaite effectuer une recherche dans FixMe, celle-ci exécute l'opération `lookup`. Cette opération décrit l'utilisation de la table de routage afin d'acheminer la requête de `lookup` de proche en proche, de la source à la destination. L'entité utilise sa table de routage afin de contacter le seed le plus proche de la destination présent dans la table de routage. Si celui-ci ne gère pas la cellule contenant la destination il transfert à son tour la requête de recherche au seed le plus proche de la destination présent dans la table de routage. Ce processus est répété successivement de proche en proche jusqu'à atteindre le seed gérant la cellule contenant la destination. Ce routage est une généralisation du routage employé dans Chord [SMK+01] à un espace de dimension d. Il nécessite donc de contacter $\mathcal{O}(d \log n)$ seeds afin d'atteindre la destination.

opération `join` Lorsqu'une entité p entre dans le système, celle-ci doit trouver le bucket auquel elle va appartenir. Elle contacte une entité q déjà présente dans le système. Cette entité q effectue alors une recherche dans le système sur la position $p_k(p)$ afin de déterminer le seed S responsable de la cellule dans laquelle p sera inséré. Le seed

S insère alors p dans le bucket approprié et met à jour les informations de sa table de routage. Dans le cas où ce bucket devient un seed (*i.e.*, plus de γ entités sont présentes dans le bucket), la cellule gérée par le seed est découpée en deux nouvelles cellules. Le pseudo-code de cette opération est fournit dans l'algorithme 8.

Algorithme 8 : p.join(t,q=None)

```
1  début
2      si q = None alors
3          q ← getBootstrapNode() ;
4      fin
5      seed ← q.lookup(Q(p,t));
6      si p ∈ seed alors
7          seed.insert(p);
8      sinon
9          bucket ← seed.findBucket(p);
10         bucket.insert(p);
11         si | bucket |≥ S_min alors
12             cells ← seed.split(bucket);
13         fin
14     fin
15 fin
```

opération `split` Suite à l'insertion d'une nouvelle entité dans un bucket, il peut arriver que celui-ci devienne un seed. La cellule contenant ce bucket peut alors détenir deux seeds S_1, S_2 contredisant ainsi la définition 12. Dans ce cas, cette cellule doit être scindée en deux nouvelles cellules, chacune de ces nouvelles cellules possèdant un seed. L'opération de `split` gère la création de ces deux nouvelles cellules.

On note $p_k(S_1) = (x_1^{(1)}, \ldots, x_d^{(1)})$ et $p_k(S_2) = (x_1^{(2)}, \ldots, x_d^{(2)})$ la position des centres respectifs de S_1 et S_2. De plus, on définit

$$i_0 = \underset{1 \leq i \leq d}{\operatorname{argmax}} \|x_i^{(1)} - x_i^{(2)}\|.$$

La dimension i_0 correspond alors à la dimension pour laquelle l'écart entre $p_k(S_1)$ et $p_k(S_2)$ est le plus important. L'hyperplan orthogonal à i_0 et passant par le point $b = (b_1, \ldots, b_d)$ définit alors la frontière entre les deux nouvelles cellules. On a :

$$\forall 1 \leq i \leq d, b_i = \begin{cases} \lfloor \frac{x_{i_0}^{(1)} + x_{i_0}^{(2)}}{2\rho_{i_0}} \rfloor \rho_{i_0} & \text{si } i = i_0 \\ 0 & \text{sinon} \end{cases}$$

Les seeds de S_1 et S_2 mettent à jour leurs tables de routage afin que chaque seed conserve les éléments appartenant à la zone qu'il gère et que l'ancienne cellule soit

remplacée par l'une des deux nouvelles créées. Les entités de S_1 et S_2 notifient ensuite les prédécesseurs de la cellule de ce changement topologique.

Exemple 19 *La figure 5.4 illustre cette opération. L'espace E est initialement partitionné en 4 cellules (figure 5.4(a)). Les cellules 1, 2 et 3 ont un seed tandis que la cellule 4 en est dépourvue. Une entité de FixMe se déplace et rejoins un nouveau bucket de la cellule A. Celui-ci devient alors un seed, la cellule 1 a donc deux seeds (figure 5.4(b)). Le seed S_1 a pour coordonnées $p_k(S_1) = (13/32, 17/32)$ et le seed S_2 $p_k(S_2) = (9/32, 7/32)$. On a $p_k(S_1) - p_k(S_2) = (4/32, 10/32)$ donc $i_0 = 2$. La frontière séparant les deux nouvelles cellules est orthogonale à l'axe des ordonnées et passe par le point de coordonnées $(0, 12/32)$. Cette frontière est illustrée par la figure 5.4(c). Finalement, les deux nouvelles cellules (cellule 5 et cellule 6) sont crées à partir de la cellule 1 (figure 5.4(d)). Les seeds de chaque cellule notifient ensuite les prédécesseurs de la cellule 1 de la création des cellules 5 et 6.*

L'algorithme 9 décrit le pseudo-code utilisé pour cette opération

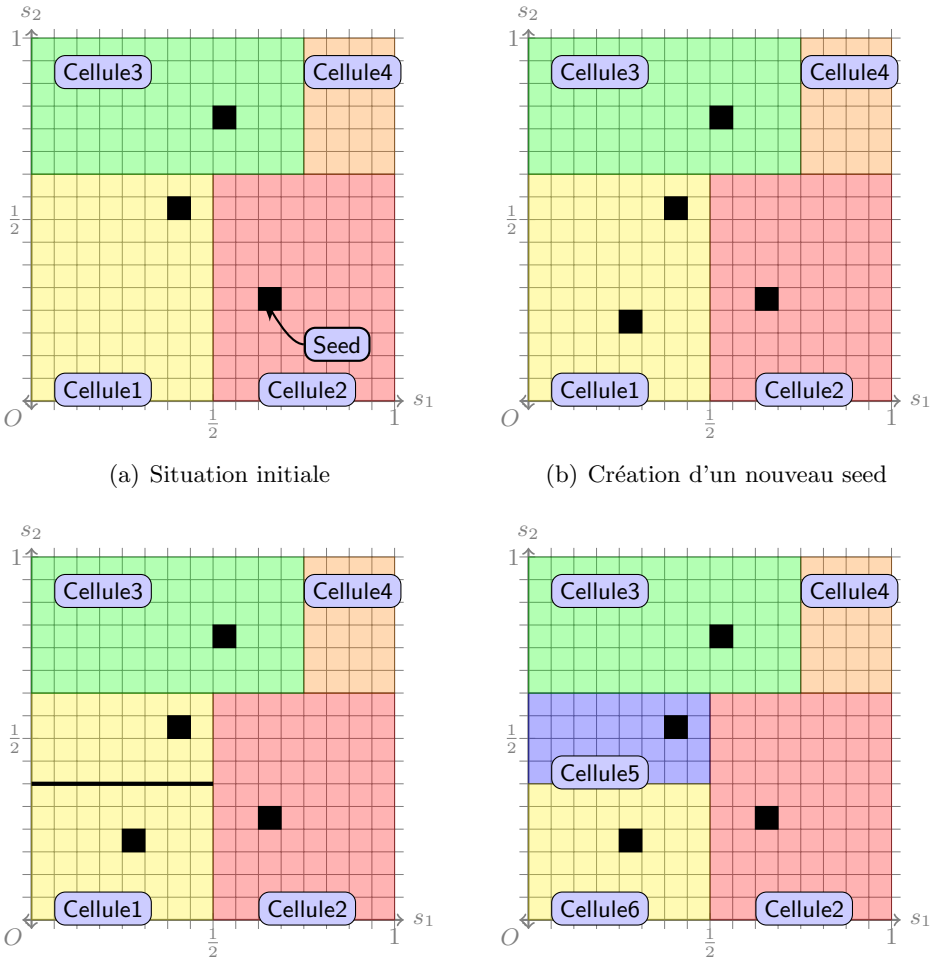
Algorithme 9 : cell.split(bucket)

Entrées : $|\ bucket\ | \geq S_{min} \wedge \neg bucket.isSeed()$
Sorties : bucket.`isSeed()`

```
1  début
2  |  matchingCell ← findCell(bucket);
3  |  si matchingCell ∈ orphanCells alors
4  |  |  bucket.cell ← matchingCell;
5  |  |  orphanCells.remove(matchingCell);
6  |  sinon
7  |  |  matchingCell.split(bucket);
8  |  fin
9  |  bucket.notifyPredecessors(matchingCell);
10 |  bucket.updateRoutingTable();
11 |  bucket.setSeed(True);
12 fin
```

opération `leave` Lorsque les mesures de performances effectuées par une entités j varient, celle-ci peut être amenée à se déplacer dans E et ainsi à devoir changer de bucket dans FixMe. Nous décrivons à présent la gestion du départ de cette entité d'un bucket de FixMe : il s'agit de l'opération `leave` dont le pseudo-code est décrit dans l'algorithme 10. Dans ce cas, le seed efface cette entité de celles présentes dans la cellule qu'il gère.

Suite au départ d'une entité d'un seed, il peut arriver que ce seed ne soit plus suffisamment peuplé. Dans ce cas, le seed redevient un bucket. La cellule contenant ce bucket n'a plus de seed. Il est nécessaire de réallouer la gestion de celle-ci à un autre seed. Cette restructuration repose sur la présence d'un *hook*.



(a) Situation initiale

(b) Création d'un nouveau seed

(c) Création de la frontière

(d) Création des cellules

FIGURE 5.4 – Déroulement d'une opération `split` dans FixMe.

Algorithme 10 : p.leave()

```
1  début
2  |  bucket ← p.bucket;
3  |  isSeed ← bucket.isSeed();
4  |  bucket.removePeer(p);
5  |  si isSeed ∧ | bucket |< S_min alors
6  |  |  bucket.setSeed(False);
7  |  |  cell ← bucket.cell.getHook();
8  |  |  cell.merge(bucket.cell);
9  |  fin
10 fin
```

Définition 13 (Hook) *Soit C une cellule de FixMe. Chaque coin de cette cellule a $2D$ buckets voisins. Le* hook *de C est le premier de ces buckets dans l'ordre lexicographique qui n'appartient pas à la cellule C.*

Proposition 2 *Pour toute cellule différente de E, le hook existe et est unique.*

Preuve Soit C une cellule de FixMe. Par définition, celle-ci est caractérisée par 2^D coins. Chaque coin a $2D$ buckets voisins. On désigne par $\mathcal{B}$ l'ensemble des buckets voisins d'un des coins de la cellule C appartenant à une cellule différente. Si $C = E$ alors tous les buckets du système appartiennent à C et on a alors $\mathcal{B} = \emptyset$. Dans le cas contraire, C a au moins une cellule voisine, et donc $\mathcal{B} \neq \emptyset$. De plus, les éléments de $\mathcal{B}$ sont totalement ordonnés suivant l'ordre lexicographique. $\mathcal{B}$ étant non vide, il existe donc nécessairement un premier élément dans cet ensemble. Il s'agit du hook et donc celui-ci est unique. □

Exemple 20 *La figure 5.5(a) illustre tous les éléments de FixMe nécessaires aux différentes opérations. Les cellules sont représentées en couleurs, les seeds correspondent au carrés noirs parallèles aux axes et les hooks aux triangles noirs.*

Considérons la cellule 3 *et supposons que son seed soit peuplé de γ entités. Lors du départ d'une de ces entités, le seed devient sous-peuplé et redevient donc un bucket. La cellule* 3 *est alors dépourvue de seed, elle est* orpheline*. Il faut donc qu'un autre seed gère cette cellule. Celui-ci est déterminé par le hook. Dans le cas présent, le hook de la cellule* 3 *appartient à la cellule* 1*. Par conséquent, le seed de la cellule* 1 *va gérer temporairement la cellule* 3*. On se retrouve alors la situation décrite dans la figure 5.5(b).*

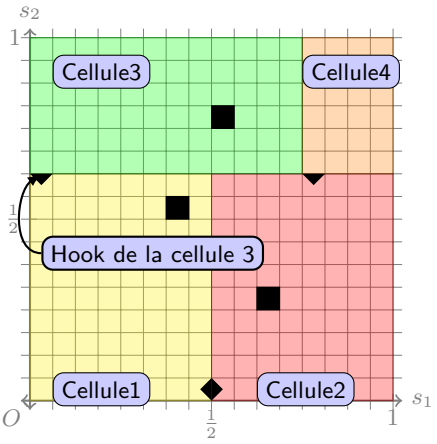
Considérons à présent la cellule 2 *de la figure 5.5(a) et supposons que son seed soit peuplé de γ entités. De la même manière que précédemment, lors du départ d'une de ces entités, le seed devient sous-peuplé, et la cellule* 2 *devient orpheline. Le hook de la cellule* 2 *appartient à la cellule* 1*, le seed de la cellule* 1 *va donc gérer temporairement la cellule* 2*. À la différence du cas précédent, les cellules* 1 *et* 2 *sont* jumelles*. Cela signifie qu'elles partagent une face complète. Dans ce cas, les deux cellules sont fusionnées en une seule cellule comme illustré par la figure 5.5(c). De plus, dans le cas présent, la cellule* 2 *gérait la cellule* 4 *orpheline. Celle-ci sera alors gérée par la nouvelle cellule* 1*.*

Finalement, dans les deux cas, la cellule prenant en charge les cellules orphelines notifie les prédécesseurs de ces cellules du changement topologique.

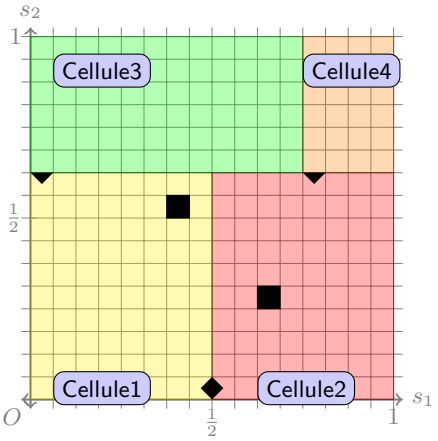
L'algorithme 11 fournit le pseudo-code de l'opération `merge`.

5.3 Gestion interne des seeds

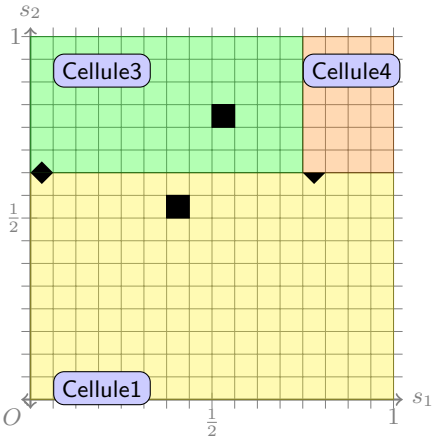
FixMe partitionne dynamiquement l'espace E en cellules en fonction de la répartition des entités dans E. Comme l'illustre la figure 5.6, le découpage de l'espace en buckets assure ainsi un équilibrage de la taille minimale des cellules. Cet équilibrage apporte



(a) Hooks des différentes cellules.



(b) Premier cas de l'opération `merge`



(c) Second cas de l'opération `merge`

FIGURE 5.5 – Utilisation des hooks dans différents cas d'exécution de l'opération `merge`.

Algorithme 11 : cell.merge(bucket)

Entrées : bucket tel que $|\ bucket\ | < S_{min}$

```
1 début
2 |   seed ← cell.seed;
3 |   seed.addOrphanCell(bucket.cell);
4 |   seed.mergeSiblingsCells();
5 |   seed.notifyPredecessors(bucket.cell);
6 fin
```

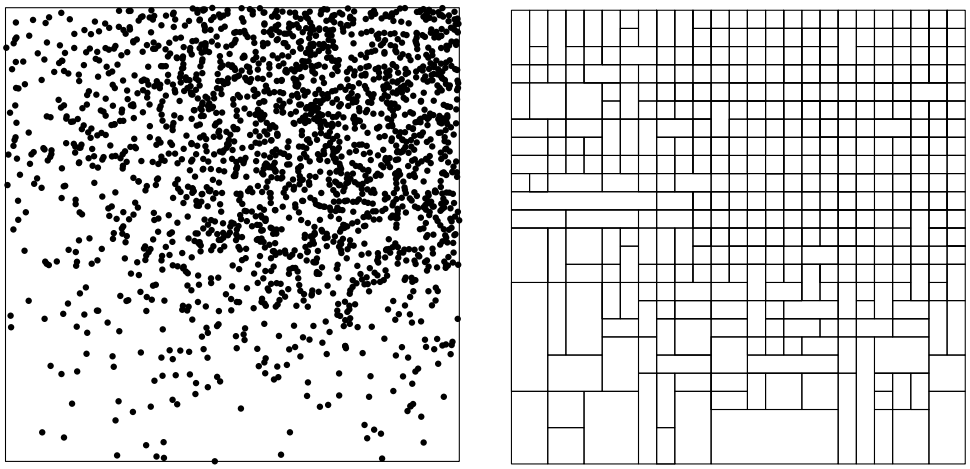


FIGURE 5.6 – Distribution non-uniforme des nœuds (à gauche) et topologie FixMe induite (à droite).

une régularité dans le partitionnement et permet ainsi une recherche plus efficace que dans le cas de CAN. De plus, l'utilisation des cellules et des seeds assure la connectivité de la topologie et donc le routage dans celle-ci.

Cependant, comme nous l'avons déjà dit, la distribution des mesures de qualités effectuées par les entités supervisées n'est pas uniforme conduisant ainsi à la création de cellules de taille variable. De plus, dans le contexte des systèmes large-échelle, il se crée des zones de l'espace très fortement peuplées. Ces zones se divisent alors en cellules de taille de plus en plus petites. Lorsque la cellule ne couvre plus qu'un seul bucket (qui est donc un seed) il n'est plus possible de la scinder en deux. Il faut alors trouver une solution pour gérer la population potentiellement grande des entités situées dans cette cellule.

Afin de gérer cette population tout en conservant les propriétés de scalabilité de FixMe, nous organisons les entités présentes au sein du seed dans une DHT. La plupart des DHT présentes dans la littérature peut convenir dans ce cas. Lors du choix de celle-ci, il faut cependant garder à l'esprit deux contraintes liées à FixMe. Tout d'abord, les entités présentes dans un seed effectuent des mesures de performance similaires. Cela signifie en particulier que des entités situées au même endroit au niveau de la topologie réseau seront situées dans le même bucket et seront susceptibles de percevoir les mêmes défaillances réseau. De ce fait, il n'est donc pas raisonnable d'utiliser une DHT conservant une localité géographique comme eQuus [LSW06]. En effet, dans le cas d'une faute massive impactant un équipement réseau, un grand nombre d'entités seront amenées à percevoir une variation anormale des mesures effectuées et devront donc se replacer dans la topologie FixMe. Si celles-ci ont en plus un placement très fortement corrélé dans la DHT sous-jacente, cela peut engendrer un grand nombre de mises-à-jours de cette topologie, voire amener à une incohérence au niveau de celle-ci.

De manière plus générale, il est nécessaire de choisir une DHT résiliente au churn,

c'est-à-dire une structure capable de gérer le dynamisme de ses entités en garantissant une relative stabilité vis-à-vis de celui-ci. Nous avons fait le choix dans le cadre de FixMe d'utiliser PeerCube [ABLR08]. Il s'agit d'une DHT résiliente à la fois au churn du système ainsi qu'aux défaillances byzantines. Sa structure absorbe les effets liés au dynamisme des entités dans le système et assure la connectivité de la topologie. De plus, ces propriétés sont assurées malgré la présence d'entités ayant un comportement arbitraire dans le système. Ces propriétés reposent sur la structure d'hypercube et l'organisation des entités au sein de clusters.

L'hypercube est une structure topologique offrant trois propriétés intéressantes. Tout d'abord, dans un hypercube ayant h sommets, le degré des sommets ainsi que le diamètre de l'hypercube vaut $d = \mathcal{O}(\log h)$. Le diamètre de l'hypercube correspond aussi à sa dimension. De plus, la construction d'un hypercube se fait facilement de manière récursive. En effet, étant donné un hypercube de dimension d il est très simple d'en construire un nouveau de dimension $d+1$: il suffit alors de dupliquer l'hypercube de dimension d et de relier ensemble les sommets identiques. On obtient alors un hypercube de dimension $d + 1$. Enfin, d'un point de vue du routage, un hypercube de dimension d fournit d chemins indépendants de longueur d entre deux sommets de l'hypercube.

Comme la plupart des DHTs, PeerCube attribue à chaque entité un identifiant unique issu d'une fonction de hachage $\mathfrak{h}$. Les entités partageant un préfixe commun dans leurs identifiants s'organisent alors au sein de clusters. Ce préfixe est appelé label du cluster. De plus, un cluster PeerCube est un ensemble d'entités dont la taille est comprise entre γ et Γ. L'ensemble des clusters forme alors une partition de l'espace d'adressage de PeerCube. Les clusters sont ensuite connectées entre eux de sorte à créer une topologie hypercubique.

Afin de limiter les coûts de maintenance de cette structure et de limiter l'impact du churn sur celle-ci, seul un sous-ensemble des entités du cluster, appelé *core*, maintient les informations nécessaires à la connectivité de l'hypercube et assure la gestion des entités du cluster. Cet ensemble est de taille minimale γ et forme une clique. Les entités du cluster n'appartenant pas au core forment le *spare*. Les entités du spare sont connectées aux entités du core et sont invisibles du reste du système.

Lorsqu'une entité rejoint un cluster, elle est ajoutée au spare. De manière similaire, lorsqu'une entité quitte le cluster, si celle-ci provient du spare, son départ est transparent du point de vue du reste du système. Le spare joue alors le rôle d'un réservoir d'entités permettant la diminution des messages de mise à jour nécessaires au maintien de la cohérence de la topologie et absorbe ainsi en partie les mises à jours dues au dynamisme des entités. À l'inverse, lors du départ d'un membre du core, une procédure de renouvellement du core est exécutée.

Les clusters se créent (opération `split`) et fusionnent (opération `merge`) suivant l'arrivée et le départ des entités de manière similaire aux cellules de FixMe. Les opérations de PeerCube (`lookup`, `join`, `leave`, `split`, `merge`) offrent toutes une complexité $\mathcal{O}(\log h)$, où h désigne le nombre d'entités dans l'hypercube.

La petite taille du core permet la gestion des entrées / sorties au niveau du cluster au moyen d'algorithmes de consensus tolérants au byzantins. Ce type d'algorithme permet alors d'assurer la mise-à-jour des tables de routage et de la composition des entités du

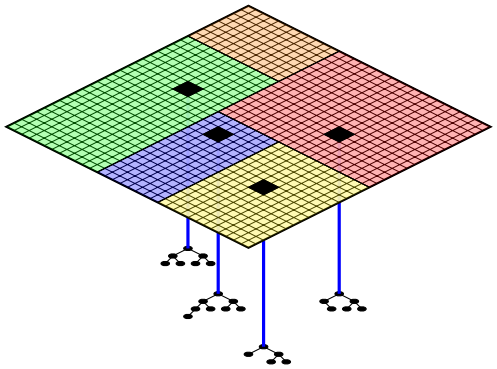


FIGURE 5.7 – Les deux niveaux de FixMe : l'espace des qualités partitionné en cellules selon la présence de seed et les DHT gérant la population de chaque seed.

cluster malgré la présence d'entités au comportement arbitraire. Une étude détaillée de la politique de renouvellement des entités formant le core ainsi que l'impact du churn sur PeerCube sera décrite dans les chapitre 6 et 7.

L'utilisation de PeerCube au niveau des seeds va permettre de gérer la population présente dans le seed. Ainsi, dans FixMe, lorsqu'une entité est ajoutée à un bucket amenant à la création d'un seed, une cellule dans l'espace des qualités E peut-être créée dans le cas où cette cellule contient déjà un seed. À la création d'un seed, un cluster PeerCube est créé. Ce cluster constitue l'instance initiale de PeerCube permettant de gérer la population de ce seed. Cette instance gère la population au fur et à mesure des entrées et des départs d'entités dans ce seed.

Le cluster contenant la clé 0 dans l'espace d'adressage PeerCube est appelé *racine* de l'hypercube. Seuls les éléments du core de la racine de l'hypercube sont visibles des autres seeds de FixMe, limitant ainsi les mises à jours nécessaires au maintien de la topologie de FixMe. La figure 5.7 illustre la décomposition de FixMe en deux niveaux. On y voit l'espace E partitionné en cinq cellules. Quatre seeds sont présents dans le système et sous chaque seed, une instance de PeerCube est présente.

5.4 Analyse

La répartition des entités dans FixMe se fait dans chacun des niveaux de l'overlay. Il se crée alors naturellement un compromis entre l'équilibrage de la population dans chacun de ces niveaux. Nous allons considérer dans cette section deux distributions pour l'analyse de la complexité des opérations.

On considère tout d'abord une distribution uniforme des n entités du système sur E. Celle-ci engendre naturellement un grand nombre de seeds. En effet, l'espace E est partitionné en $\prod_{i=1}^{d} b_i$ buckets. Dans le cas où $n \geq \gamma \prod_{i=1}^{d} b_i$, en moyenne, tous les buckets

du système sont des seeds et chaque seed contient alors $n/\prod_{i=1}^{d} b_i$ entités.

À l'inverse, lorsque les entités son concentrées sur une unique position dans l'espace des qualités, on n'a qu'un seul seed dans le système. Toutes les entités appartiennent alors à la même instance PeerCube. Dans ce cas, on a alors n entités situées dans le même seed.

Ces deux distributions représentent les cas limites sur le nombre de seeds dans le système et le nombre d'entités dans une instance PeerCube. Nous allons étudier dans cette section la complexité des différentes opérations nécessaires à FixMe avec ces deux distributions de positions des entités.

Proposition 3 (Opération `join`) *Lorsqu'une entité est insérée dans un bucket sans changement topologique, la complexité de l'opération est* $\Theta(\log h)$ *avec* h *le nombre d'entités dans l'hypercube sous-jacent.*

Preuve Lors d'une insertion sans changement topologique, la complexité de l'opération est déterminée par celle de l'insertion dans l'hypercube sous-jacent. Celle-ci vaut $\Theta(\log h)$ avec h le nombre d'entités déjà présentes dans celui-ci. □

Proposition 4 (Opération `split`) *Lorsque l'insertion d'une entité dans un bucket induit un changement topologique dans l'espace* E*, la complexité de l'opération vaut* $\mathcal{O}(d)$*.*

Preuve Comme nous l'avons vu dans la section précédente, l'ajout d'une entité dans un bucket peut entraîner un changement topologique au niveau de l'espace des qualités dans le cas où ce bucket devient un seed. Cette opération nécessite une unique écriture dans la table de routage des éléments du seed s_1 gérant la cellule. Le nouveau seed quant à lui crée une nouvelle instance de PeerCube avec les γ entités présentes en temps constant. De plus, il lui faut créer une nouvelle table de routage pour l'espace des qualités. Cette création engendre l'envoi de $2\sum_{i=1}^{d} \lfloor \log_2(R_i/\rho_i) \rfloor - 1$ requêtes de lookup. □

Proposition 5 (Opération `leave`) *La suppression d'une entité sans changement topologique requière l'envoi de* $\Theta(\log(h))$ *messages, avec* h *le nombre d'entités présentes dans l'hypercube sous-jacent.*

Preuve Lorsqu'une entité quitte un bucket, elle est simplement supprimée de son cluster dans l'hypercube sous-jacent. On distingue deux cas : la quantité d'entités restante dans le cluster est suffisante et dans ce cas, les entités du cluster mettent à jour la vue de celui-ci, ou alors le cluster n'est plus suffisamment peuplé et dans ce cas le cluster doit fusionner avec un autre, engendrant ainsi l'envoi de $\Theta(\log(h))$ messages

(voir [ABLR08]). □

On considère un seed S tel que $|S| = \gamma$. Le départ d'une entité de ce seed le ramène à l'état de bucket. Dans ce cas, la cellule C gérée par S passe sous la responsabilité du seed gérant la cellule contenant le hook de C. Il est alors nécessaire de notifier les prédécesseurs de C de ce changement topologique. La proposition suivante permet de déterminer une borne supérieure sur ce nombre de prédécesseurs.

Proposition 6 *Soient $C \subseteq E$ une cellule de E et p_i le nombre de seeds ayant C à la i-ème entrée de leur table de routage, on a :*

$$p_i \leq 2\log_2(b_i/2) \prod_{j=1, j\neq i}^{d} b_j,$$

où $b_i = 1/\rho_i$ désigne le nombre d'intervalles élémentaires de la dimension i.

Preuve Soit ℓ_i la longueur du coté de C suivant la i-ème dimension. Dans le cas ou chaque bucket voisin de la cellule C est un seed, C a alors au plus $\prod_{j=1, j\neq i}^{d} \ell_j/\rho_j$ cellules voisines sur chaque face. On a donc $p_i \leq 2K^{(i)} \prod_{j=1, j\neq i}^{d} \ell_j/\rho_j$.

Comme nous l'avons vu dans la proposition 1, on a $K^{(i)} = \lfloor \log_2(R_i/\rho_i) \rfloor$. De plus, l'espace des qualités E ayant une structure torique et étant muni de la norme infinie, on a $\forall (j, \ell) \in E^2, \|j - \ell\| \leq 1/2$. On a donc $\forall i \in \{1, \ldots, d\}$ on a $R_i \leq 1/2$. Enfin, $E = [0, 1]^d$, on a donc $\forall i \in \{1, \ldots, d\}, \ell_i \leq 1$. Par conséquent, on a $p_i \leq 2\log_2(b_i/2) \prod_{j=1, j\neq i}^{d} b_j$. □

Proposition 7 (Opération `merge`) *Lorsqu'un cellule doit fusionner avec une autre, cette opération engendre au plus $2db^{d-1}\log_2(b/2)$ messages, avec $b = \max_{1\leq i\leq d} b_i$.*

Preuve On considère la cellule C_1 gérée par le seed S_1 tel que $|S_1| = \gamma$. Suite au départ d'une entité de S_1, on a $|S_1| < \gamma$. Le seed S_1 redevient alors un bucket et la cellule C_1 n'est lus gérée, l'opération `merge` est exécutée. Les entités de S_1 contactent alors le seed S_2 en charge de la cellule C_2 contenant le hook de C_1. Les entités de S_2 notifient alors les prédécesseurs de S_1.

D'après la proposition 6, le nombre de prédécesseur $p = \sum_{i=1}^{D} p_i$ satisfait l'inégalité suivante :

$$p \leq 2\sum_{i=1}^{d} \log_2(b_i/2) \prod_{j=1, j\neq i}^{d} b_j.$$

On note $b = \max_{1 \leq i \leq d} b_i$. On a alors $b^{d-1} \geq \prod_{j=1, j \neq i}^{d} b_j$.

Le nombre de prédécesseurs vérifie alors :

$$p \leq 2b^{d-1} \sum_{i=1}^{d} \log_2(b_i/2).$$

D'autre part, on a $\forall 1 \leq i \leq d, \log(b) \geq \log(b_i)$. On a donc $p \leq 2db^{d-1} \log_2(b/2)$. □

Proposition 8 (Opération `lookup`) *Pour les deux distributions de positions d'entités sur E considérées, une opération de `lookup` engendre $\mathcal{O}(\log n)$ messages, avec n représentant le nombre d'entités présentes dans le système.*

Preuve Supposons tout d'abord que les mesures de qualités effectuées par les entités du systèmes soient uniformément distribuées sur E. Dans ce cas, les entités sont insérées dans chaque bucket de manière uniforme. Ainsi, cette distribution maximise le nombre de seeds dans l'espace des qualités E et minimise le nombre moyen d'entités h dans chaque hypercube. Pour un nombre n d'entités suffisamment grand, tous les buckets sont des seeds et on a alors $\prod_{i=1}^{d} b_i$ seeds dans E et chaque seed gère alors une zone égale au seed. D'autre part, on a $b_i = 1/\rho_i$ ce qui nous donne $h = n \prod_{i=1}^{d} \frac{1}{\rho_i}$.

Comme nous l'avons décrit dans la section 5.2, l'opération de `lookup` est décomposée en trois phases : traversée de l'hypercube de la cellule source, navigation dans l'espace des qualités et enfin traversée de l'hypercube de la cellule destination. Comme il a été montré dans [ABLR08], la dimension de l'hypercube contenant h entités vaut $\log h/\Gamma$. La dimension de l'hypercube correspond aussi à son diamètre. Le nombre de messages requis pour traverser l'hypercube vaut donc $\log h/\Gamma$.

Dans le cas $\Gamma = \log n$, le nombre de messages requis pour traverser l'hypercube vaut

$$\log(n \prod_{i=1}^{d} \frac{1}{\rho_i}) - \log\log n = \log(n) - \log(\prod_{i=1}^{d} \frac{1}{\rho_i}) - \log\log n = \mathcal{O}(\log n).$$

D'autre part, la navigation dans l'espace des qualités nécessite $\mathcal{O}(d \log n)$ messages. Ainsi, dans le cas où les entités sont uniformément distribuées sur E, le nombre total de messages requis pour une opération de `lookup` vaut $\mathcal{O}(\log n)$.

Supposons à présent que toutes les entités du système mesurent, pour chaque service, la même valeur de qualité. Toutes les entités ont alors la même position dans E et donc appartiennent au même bucket. Dans le cas où $n \geq \gamma$, il existe une unique cellule dans le système et toutes les entités appartiennent au même hypercube sous-jacent. Dans ce cas, sa dimension est maximale. De manière similaire au cas précédent,

TABLE 5.1 – Liste des notations

Notation	Signification
n	Nombre d'entités du système
E	Espace des qualités (Section 3.1.1)
d	Dimension de l'espace des qualités
ρ_i	Longueur de l'intervalle élémentaire dans la dimension i
b_i	Nombre d'intervalles élémentaires dans la dimension i
R_i	Distance maximale atteignable en un saut dans la dimension i
$p_k(j)$	Position dans E à l'instant k de l'entité j (Section 3.1.2)
γ	Taille minimale d'un seed dans FixMe, et d'un cluster de PeerCube
Γ	Taille maximale d'un cluster de PeerCube
r	Rayon de cohérence (Section 3.1.1)
τ	Seuil de densité (Définition 6)
$a_k(j)$	Détection d'anomalie par l'entité j à l'instant k (Définition 7)
$\mathfrak{h}$	Fonction de hachage utilisée pour déterminer les identifiants dans PeerCube
$\mathfrak{B}(o, r)$	Boule de rayon r centrée sur le point o
$\overline{\mathcal{W}}_k(j)$	Famille des ensembles contenant l'entité j ayant un mouvement τ-dense maximal dans l'intervalle de temps $[k-1, k]$ (Section 3.3)
$J_k(j)$	Ensemble des entités de $D_k(j)$ pour lesquelles j appartient à tous les mouvements τ-dense maximaux dans l'intervalle de temps $[k-1, k]$ (Section 3.3)
T	Délai nécessaire aux entités pour se replacer dans FixMe et écrire les données nécessaires aux algorithmes de détection

on montre que le nombre total de message requis pour une opération de `lookup` vaut $\mathcal{O}(\log n)$ dans ce cas. □

Nous venons de présenter dans les sections précédentes l'architecture de FixMe ainsi que la complexité algorithmique de ses opérations. Dans la section suivante nous nous intéressons à l'utilisation de cette architecture en vue de déterminer pour chaque entité supervisée percevant une défaillance, le type de faute l'ayant provoquée.

5.5 Utilisation de FixMe pour la caractérisation de fautes

La caractérisation de fautes présentée dans les chapitres 3 et 4 repose sur la possibilité de construire pour chaque entité $j \in [\![1, n]\!]$ percevant une défaillance la famille $\overline{\mathcal{W}}_k(j)$ des ensembles ayant un mouvement τ-dense maximal dans l'intervalle de temps $[k-1, k]$. Ainsi, lorsqu'une entité $j \in [\![1, n]\!]$ perçoit une défaillance, il est nécessaire de déterminer l'ensemble des entités percevant une défaillance similaire, c'est-à-dire

les entités dont les variations de mesures de qualités sont fortement corrélés à celles mesurées par j. Rappelons la définition de cet ensemble noté $N(j)$:

$$N(j) = \{\ell \in [\![1, n]\!] \setminus \{j\} \mid \|p_{k-1}(j) - p_{k-1}(\ell)\| \leq 2r, \|p_k(j) - p_k(\ell)\| \leq 2r\}.$$

L'algorithme 12 décrit comment les entités supervisées utilisent FixMe. Les entités de FixMe effectuent des mesures de performances des services consommés à des instants discrets. Ces mesures ne sont pas nécessairement stables dans le temps et il peut arriver qu'il y ait des variations dans ces mesures sans que celles-ci soient considérées comme des anomalies. Lorsque les mesures effectuées par l'entité j supervisée diffèrent des précédentes mesures effectuées, celle-ci se déplace dans FixMe et peut donc être amenée à devoir changer de bucket. Ce déplacement consiste à déterminer le nouveau bucket auquel l'entité j appartient. Si ce bucket appartient à une autre cellule, l'entité j quitte sa cellule courante afin d'être insérée dans sa nouvelle cellule. La tâche 1 de l'algorithme 12 décrit le pseudo-code nécessaire à ce déplacement.

Dans le cas où l'entité j perçoit une défaillance, la fonction de détection d'anomalie $a_k(j)$ vaut `Vrai`. L'entité cherche alors à déterminer le type de faute ayant entraîné cette défaillance. Pour cela, j construit l'ensemble $N(j)$ afin de déterminer la famille $\overline{\mathcal{W}}_k(j)$. Cet ensemble correspond à l'intersection des voisinages $N_{k-1}(j)$ de j à l'instant $k-1$ avec son voisinage $N_k(j)$ à l'instant k.

Nous utilisons les instances de PeerCube présentes dans chaque cellule comme une mémoire partagée où chaque entité percevant une défaillance peut écrire et lire. La lecture (opération `get`) et l'écriture (opération `put`) dans l'hypercube se fait à l'adresse $\mathfrak{h}((B, k))$, où $\mathfrak{h}$ est la fonction de hachage utilisée par PeerCube pour le placement des entités, B correspond à un bucket et k à l'instant discret considéré.

La tâche 2 de l'algorithme 12 consiste alors à écrire le triplet $(j, p_{k-1}(j), p_k(j))$ à différentes adresses $\mathfrak{h}((B, k))$ dans l'hypercube sous-jacent. Ces adresses correspondent à chaque bucket B de FixMe ayant une intersection non vide avec la boule $\mathfrak{B}(p_k(j), 4r)$ de rayon $4r$ centrée sur la position de j à l'instant k.

Après un délai T, suffisamment grand pour que chaque entité percevant une défaillance se soit replacée dans FixMe et ait exécuté la tâche 2, l'entité j récupère les informations stockées aux adresses $\mathfrak{h}((B_k, k))$ dans l'hypercube. Ces informations constituent un sur-ensemble des entités de $N(j)$. On ne conserve que les entités appartenant à $N(j)$ (Tâche 3). Finalement, la tâche 4 exécute l'algorithme 3.

Théorème 7 *L'algorithme 12 détermine pour chaque entité percevant une défaillance le type de faute qui l'a provoqué.*

Preuve On montre que les tâches 2 et 3 de l'algorithme 12 construisent l'ensemble $N(j) = N_k(j) \cap N_{k-1}(j)$.

Soit $j \in [\![1, n]\!]$ une entité percevant une défaillance dans l'intervalle de temps $[k-1, k]$. L'ensemble $N_k(j)$ est défini par $N_k(j) = \{\ell \in A_k \mid \|p_k(j) - p_k(\ell)\| \leq 2r\}$. On a donc $N(j) \subseteq N_k(j) \subseteq \mathfrak{B}(p_k(j), 4r)$.

On considère à présent la famille de buckets $\mathcal{B} = \{B \subseteq E \mid B \cap \mathfrak{B}(p_k(j), 4r) \neq \emptyset\}$. Par définition de cet ensemble, on a $\mathfrak{B}(p_k(j), 4r) \subseteq \bigcup_{B \in \mathcal{B}} B$ et donc $N_k(j) \subseteq \bigcup_{B \in \mathcal{B}} B$. Les buckets de $\mathcal{B}$ contiennent donc tous les éléments de $N_k(j)$.

Par ailleurs, toutes les entités percevant une défaillance exécutent l'algorithme 12, et donc en particulier ceux de $N(j)$. L'entité j écrit à chaque adresse $\mathfrak{h}((B,k)), \forall B \in \mathcal{B}$. Par conséquent, la distance étant symétrique, toutes les entités appartenant à un bucket de B écrivent dans le bucket B_k contenant j. La donnée située à l'adresse $\mathfrak{h}((B_k,k))$ contient donc tous les triplets $(\ell, p_{k-1}(\ell), p_k(\ell)), \forall \ell \in N_k(j)$. D'après la définition de $N(j) = N_k(j) \cap N_{k-1}(j)$, on a $N(j) \subseteq N_k(j)$. Il est donc suffisant d'écrire dans ces buckets.

Cependant, comme la famille de buckets $\mathcal{B}$ constitue un sur-ensemble de $\mathfrak{B}(p_k(j), 4r)$, tous les triplets contenus à l'adresse $\mathfrak{h}((B_k,k))$ ne vérifient pas $\|p_k(j) - p_k(\ell)\| \leq 2r$. D'autre part, certains éléments ne vérifient pas la condition $\|p_{k-1}(j) - p_{k-1}(\ell)\| \leq 2r$. On ne conserve que les éléments vérifiant ces deux conditions, ce qui correspond à l'ensemble $N(j)$. Cet ensemble est fourni en paramètre de l'algorithme 3. Celui-ci applique les théorèmes permettant de déterminer le type de faute ayant provoqué la défaillance perçue par l'entité j. □

Proposition 9 *L'exécution de l'algorithme 12 nécessite $\mathcal{O}(\log n)$ messages.*

Preuve L'algorithme 12 est constitué de quatre tâches.

La tâche 1 nécessite une opération `lookup` (ligne 5) et d'une opération `join` (ligne 8). Ces deux opérations nécessitent $\mathcal{O}(\log n)$ messages d'après les propositions 3 et 8.

La tâche 2 nécessite une opération `lookup` et une opération `put` dans PeerCube par élément $B \in \mathcal{B}$. La famille $\mathcal{B}$ contient $\prod_{i=1}^{d} \lceil 4r/\rho_i \rceil$ buckets. Les opérations `lookup`, `put` dans PeerCube nécessitent chacune l'envoi de $\Theta(\log h)$ messages dans une instance de PeerCube contenant h entités. On a $h \leq n$. Par conséquent, la tâche 2 nécessite l'envoi de $\mathcal{O}(\log n)$ messages.

La tâche 3 nécessite une opération `get` (ligne 22) dans PeerCube. Cette opération a le même coût qu'une opération `lookup` dans PeerCube : $\Theta(\log h)$ messages. Comme précédemment, on a $h \leq n$, la tâche 3 nécessite donc l'envoi de $\mathcal{O}(\log n)$ messages.

Enfin la tâche 4 ne nécessite pas l'envoi de messages. On a donc un coût total de $\mathcal{O}(\log n)$ messages. □

Cet algorithme peut être simplifié dans le cas où les valeurs des mesures de qualités sont discrètes et que le rayon de cohérence r est inférieur à la subdivision de l'espace des qualités en buckets, c'est-à-dire si $r < \min_{1 \leq i \leq d} \rho_i$. Dans ce cas, l'espace des qualités E a une structure de grille torique.

De plus, la condition $r < \min_{1 \leq i \leq d} \rho_i$ implique qu'une défaillance n'impacte des entités n'appartenant qu'à un seul bucket exactement. Par conséquent, l'algorithme permettant

Algorithme 12 : j.updatePosition(k :round)

Entrées : T : délai nécessaire au replacement des entités dans FixMe.
Sorties : L'entité p est replacée dans le bucket contenant sa position actuelle. Si p perçoit une défaillance dans l'intervalle de temps $[k-1, k]$, la faute associée est caractérisée au moyen de l'algorithme 3, 4 ou 5.

début
  tâche 1
    $k \leftarrow k+1$;
    $B_k \leftarrow$ j.bucket$(p_k(j))$;
    $C_k \leftarrow$ j.`lookup(`$p_k(j)$`)`;
    si $C_k \neq C_{k-1}$ **alors**
      j.`leave()`;
      j.`join(`j`)`;
    fin
  fin
  si $a_k(j)$=*Vrai* **alors**
    tâche 2
      pour chaque $B \subseteq E, B \cap \mathfrak{B}(p_k(j), 2r) \neq \emptyset$ **faire**
        key$\leftarrow \mathfrak{h}((B,k))$;
        $C \leftarrow$ j.`lookup(`B`)`;
        j.`put(`C,key,$(j, p_{k-1}(j), p_k(j))$`)`;
      fin
    fin
    attendre pendant T;
    tâche 3
      key$\leftarrow \mathfrak{h}((B_k,k))$;
      $L \leftarrow$ j.`get(`key`)`;
      $N(j) \leftarrow \{(\ell, p_{k-1}(\ell), p_k(\ell)) \in L \mid \ell \neq j, \|p_{k-1}(j) - p_{k-1}(\ell)\| \leq 2r, \|p_k(j) - p_k(\ell)\| \leq 2r\}$;
    fin
    tâche 4
      faulttype $\leftarrow$j.`characterize()`;
    fin
  fin
fin

Algorithme 13 : j.updatePosition(k :round)

Entrées : T : délai nécessaire au replacement des entités dans FixMe.
Sorties : L'entité p est replacée dans le bucket contenant sa position actuelle. Si p perçoit une anomalie dans l'intervalle de temps $[k-1, k]$, la faute associée est caractérisée.

```
1  début
2      tâche 1
3          k ← k + 1;
4          B_{k-1} ← j.bucket(p_{k-1}(j)) ;
5          B_k ← j.bucket(p_k(j)) ;
6          C_k ← j.lookup(p_k(j));
7          si C_k ≠ C_{k-1} alors
8              j.leave();
9              j.join(j);
10         fin
11     fin
12     si a_k(j)=Vrai alors
13         tâche 2
14             key← ɦ((B_{k-1}, B_k, k));
15             C ← j.lookup(key);
16             n ←j.get(C,key);
17             j.put(C,key,n + 1);
18         fin
19         attendre pendant T;
20         tâche 3
21             n ← p.get(cell,key);
22             si n ≤ τ alors
23                 faulttype ← Isolated;
24             sinon
25                 faulttype ← Massive;
26             fin
27         fin
28     fin
29 fin
```

de déterminer si l'entité perçoit une défaillance due à une faute isolée ou à une faute massive peut être simplifié. Cette version simplifiée est fournie dans l'algorithme 13.

De manière similaire à l'algorithme 12, l'algorithme 13 est décomposé en trois tâches. Comme précédemment, dans la première tâche, l'entité supervisée effectue ces mesures de performance et se déplace éventuellement d'un bucket B_{k-1} à un autre bucket B_k pouvant nécessiter le changement de cellule.

Si la variation de mesure est perçue comme une anomalie, alors la fonction $a_k(j)$ retourne `Vrai` et les tâches 2 et 3 sont exécutées. Contrairement à l'algorithme 12, l'entité incrémente un compteur situé à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$ dans l'hypercube sous-jacent.

Après un délai T, suffisamment grand pour que chaque entité se soit replacée dans FixMe et ait exécuté la tâche 2, l'entité j récupère la valeur du compteur situé à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$. Si cette valeur est supérieure au seuil de densité τ, l'entité perçoit une défaillance due à une faute massive. Sinon, la valeur du compteur situé à cette adresse est inférieure à τ, cette défaillance est due à une faute isolée.

Théorème 8 *Si les mesures de qualité effectuées par les entités supervisées ont des valeurs discrètes dans E : $\forall 1 \leq i \leq d, q_{i,k}(j) \in \{0, \rho_i, 2\rho_i, \ldots, 1\}$ et si le rayon de cohérence r vérifie $r < \min_{1 \leq i \leq d} \rho_i$ alors l'algorithme 13 détermine pour chaque entité percevant une défaillance le type de faute qui l'a provoqué.*

Preuve Par hypothèse du théorème, les entités sont placées dans E selon une grille discrète. De plus, on a $r < \min_{1 \leq i \leq d} \rho_i$. Cela signifie donc que seules les entités situées à la même position discrète peuvent être impactées par une même faute.

Supposons qu'à l'instant k, seul un ensemble S de $m \leq \tau$ entités supervisées appartiennent au bucket B_{k-1} et perçoivent une défaillance due à une même faute isolée. Ces entités se déplacent dans un autre bucket B_k. Pour chacune de ces entités ℓ, on a $a_k(\ell) =$`Vrai`. Cet ensemble a donc un mouvement non τ-dense.

Par hypothèse de la preuve, seules ces entités se sont déplacées de B_{k-1} à B_k. Pour chaque entité $\ell \in S$, on a $\overline{\mathcal{W}_k}(\ell) = \emptyset$. Par application du théorème 2, ces entités sont impactées par une faute isolée. Par application de l'algorithme 13, chacun des entités de S écrit à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$ dans l'hypercube de la cellule contenant B_k. Après le délai T, toutes les entités de S lisent la même valeur $|S|$ à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$. On a $|S| \leq \tau$, l'algorithme 13 retourne alors *Isolated*.

Supposons à présent qu'à l'instant k, seul un ensemble S de $m > \tau$ entités supervisées appartiennent au bucket B_{k-1} et perçoivent une défaillance due à une même faute massive et se déplacent dans un autre bucket B_k. Pour chacune de ces entités ℓ, on a $a_k(\ell) =$`Vrai`. Cet ensemble a donc un mouvement τ-dense. De plus, comme on a $r < \min_{1 \leq i \leq d} \rho_i$, S est le seul ensemble ayant un mouvement τ-dense auquel appartiennent les entités de S.

On a donc $\forall \ell \in S, J_k(\ell) = S$. Par application du théorème 3, ces entités sont impactées par une faute massive. Par application de l'algorithme 13, chacun des entités de S écrit à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$ dans l'hypercube de la cellule contenant B_k. Après

le délai T, toutes les entités de S lisent la même valeur $|S|$ à l'adresse $\mathfrak{h}((B_{k-1}, B_k, k))$. On a $|S| > \tau$, l'algorithme 13 retourne alors *Massive*.

Les deux derniers cas à considérer ne respectent pas les hypothèses du modèle. En effet, le cas où $m > \tau$ entités se déplaçant de B_{k-1} à B_k perçoivent une défaillance due à des fautes isolées viole la condition **C1** de la définition 8. De la même manière, le cas où $m > \tau$ entités perçoivent une défaillance due à des fautes massives et $m' \leq \tau$ entités perçoivent une défaillance due à des fautes isolées se déplaçant de B_{k-1} à B_k viole la condition **C2** de la définition 8. □

Proposition 10 *L'exécution de l'algorithme 13 nécessite $\mathcal{O}(\log n)$ messages.*

Preuve De manière similaire à l'algorithme 12, l'algorithme 13 nécessite une opération `lookup` (ligne 6) et une opération `join` (ligne 9). Ces deux opérations nécessitent $\mathcal{O}(\log n)$ messages d'après les propositions 3 et 8.

La tâche 2 nécessite une opération `lookup`, une opération `get` et une opération `put` dans PeerCube. Chacune des opérations `lookup`, `put` et `get` dans PeerCube nécessite l'envoi de $\mathcal{O}(\log n)$ messages dans une instance de PeerCube contenant h entités.

La tâche 3 nécessite une opération `get` (ligne 21) dans PeerCube de coût $\mathcal{O}(\log n)$. On a donc un coût total de $\mathcal{O}(\log n)$ messages. □

5.6 Discussion

L'algorithme des κ-moyennes [HW79] est un algorithme de partitionnement de données basé sur la distance. L'algorithme partitionne un ensemble de points S en différents clusters en fonction d'une mesure de distance D. C'est une méthode dont le but est de diviser des observations en κ parties disjointes dans lesquelles chaque observation appartient à la partie ayant la moyenne la plus proche de l'observation considérée.

Étant donné un ensemble d'entités $S = [\![1, n]\!]$ placées dans $E = [0, 1]^d$ l'algorithme κ-moyennes partitionne les n observations en κ ensembles $\Sigma_1, \Sigma_2, \ldots, \Sigma_\kappa$ ($\kappa \leq n$). On utilise ici la mesure de distance induite par la norme infinie.

L'algorithme est initialisé avec un ensemble de κ points référents $R = \{m_1, \ldots, m_\kappa\}$. L'algorithme construit alors les ensembles suivants.

$$\Sigma_i = \{j \in S \mid \|j - m_i\| \leq \|j - m_\ell\| \forall \ell \in [\![1, \kappa]\!]\}.$$

Les points référents m_i sont replacés à la position de l'isobarycentre des entités de Σ_i. On a :

$$\forall 1 \leq i \leq \kappa, m_i = \frac{1}{|\Sigma_i|} \sum_{j \in \Sigma_i} p_k(j).$$

Ce processus est répété jusqu'à ce que les points m_i soient stables.

Exemple 21 *La figure 5.8 illustre le déroulement de cet algorithme. Le système est composé de n=35 entités consommant d=2 services. Ces entités sont placées dans l'espace des trajectoires $T = [0, 1]^2$ comme illustré sur la figure 5.8(a). L'algorithme est initialisé en choisissant uniformément $\kappa = 4$ positions pour les référents parmi celles des entités présentes dans le système. Les positions des référents sont représentées avec des carrés de couleur sur la figure 5.8(b). On associe à chacune des n entités, le point référent m_i le plus proche. Cette association construit alors les clusters représentés sur la figure 5.8(c). Les points référents sont replacés en fonction des entités de leur cluster (figure 5.8(d)). Ce processus est répété (figure 5.8(e)) jusqu'à ce que les points m_i soient stables (figure 5.8(f)).*

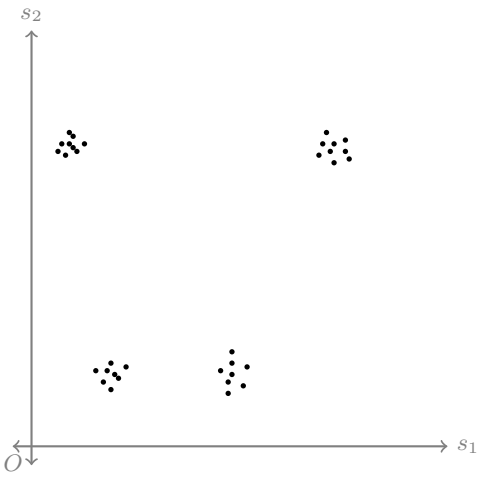
Il existe un nombre fini de partitions en κ parties d'un ensemble de n éléments. De plus, chaque itération de l'algorithme fait strictement diminuer la distance moyenne entre le point référent et les positions des entités de son cluster. L'algorithme converge donc toujours en temps fini. Cependant, l'algorithme ne converge que vers un minimum local.

La figure 5.9 illustre ce point. Étant donné le même système composé de n=35 entités consommant d=2 services aux mêmes positions, on initialise à présent l'algorithme avec $\kappa = 4$ autres entités. La figure 5.9 illustre alors le déroulement de l'algorithme dans ce cas. On constate sur la figure 5.9(f) que la partition finale diffère de celle illustrée dans la figure 5.8(f). L'algorithme des κ-moyennes est donc très sensible à son initialisation.

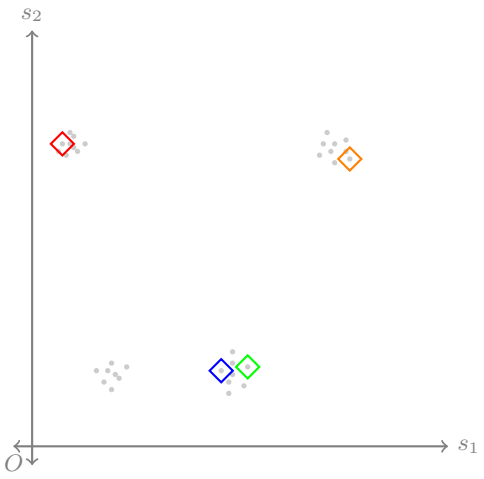
D'autre part, le paramètre κ est fondamental pour cet algorithme. La figure 5.10 illustre le résultat de cet algorithme sur le même système que précédemment. On constate que si le paramètre κ est mal positionné, l'algorithme ne réussit pas à exhiber une partition en adéquation avec le placement des entités dans l'espace des trajectoires. Il n'est pas évident de trouver une valeur de ce paramètre en fonction de la répartition des entités dans l'espace des qualités. De plus, la position des entités dans l'espace des qualités évoluant à chaque instant discret, la valeur idéale du paramètre κ est susceptible d'évoluer aussi. La figure 5.11 illustre l'évolution de la valeur idéale de κ à différents instants discrets.

Le partitionnement de l'espace des qualités en buckets utilisé par FixMe nous permet de nous affranchir du paramètre κ. Les buckets suffisamment peuplés passent à l'état de seed et se comportent alors comme les référents de l'algorithme κ-moyennes. La figure 5.12 illustre ce comportement sur le système décrit dans la figure 5.11. Le système est composé de 35 entités consommant $d = 2$ services. Les entités sont placées dans FixMe en fonction de leurs mesures de qualités. FixMe partitionne l'espace des qualités en cellules en fonction de la présence des seeds. À l'instant $k - 1$, 4 seeds sont présents dans le système et l'espace est divisé en 4 cellules. À l'instant k, 4 entités ont perçu une variation de qualité et se sont replacées dans FixMe. Un nouveau seed et une nouvelle cellule sont créées. Le partitionnement de l'espace des qualités permet à FixMe de s'adapter dynamiquement au placement des entités dans l'espace des qualités et ainsi de s'affranchir du paramètre κ présent dans l'algorithme des κ-moyennes.

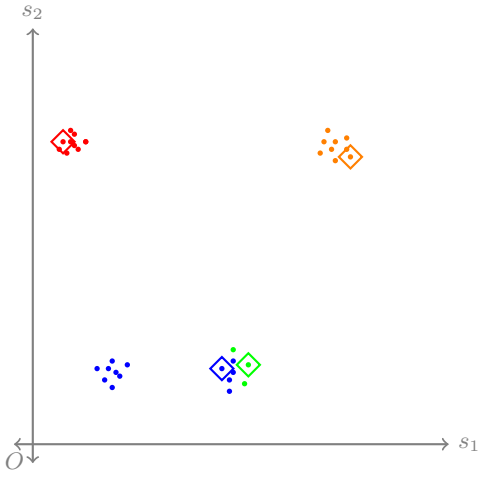
L'algorithme κ-moyennes effectue une décomposition de Voronoi de l'espace des qualités E par rapport aux positions des κ points référents. Cette décomposition assure



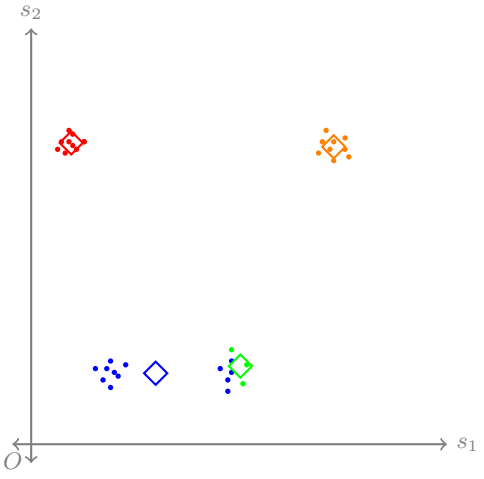
(a) Initialisation



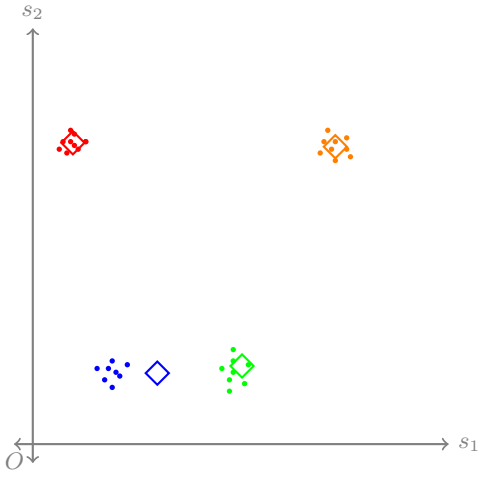
(b) Sélection des positions initiales des m_i



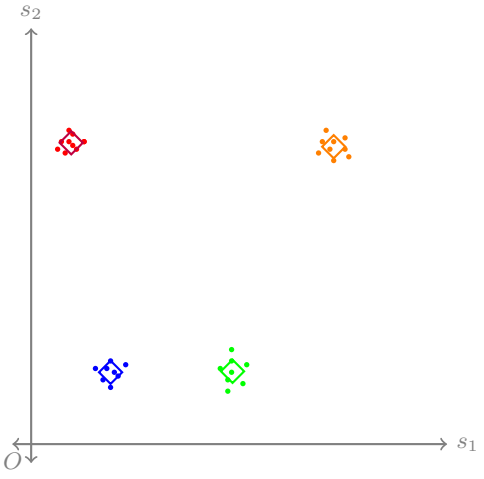
(c) Construction des clusters



(d) Mise à jour des positions des m_i

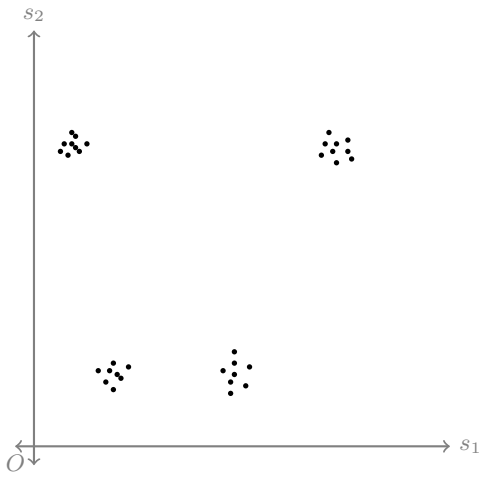


(e) Mise à jour des clusters

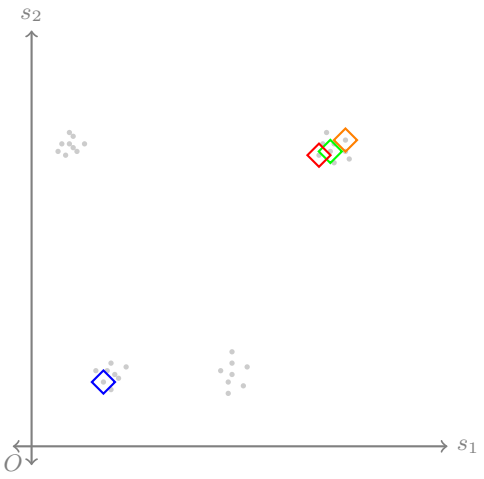


(f) Clusters finaux

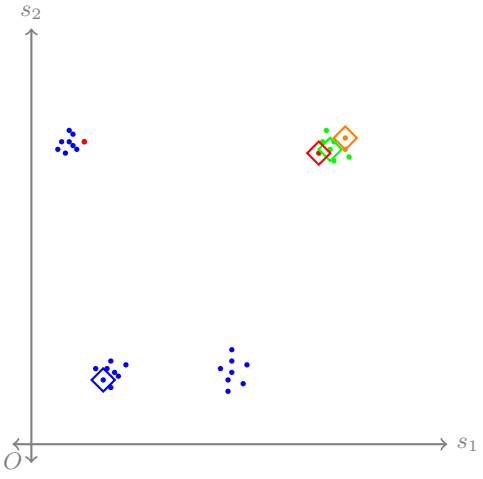
FIGURE 5.8 – Exécution de l'algorithme κ-moyennes.



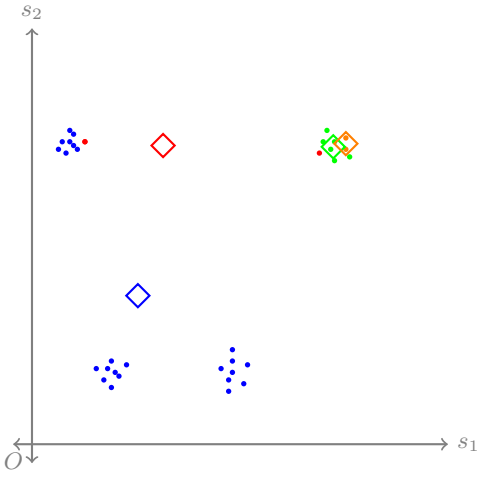
(a) Initialisation



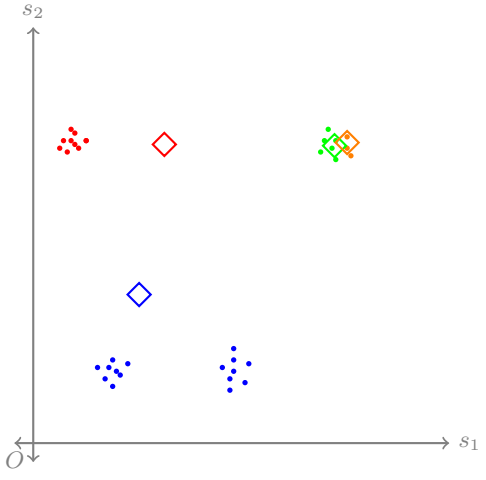
(b) Sélection des positions initiales des m_i



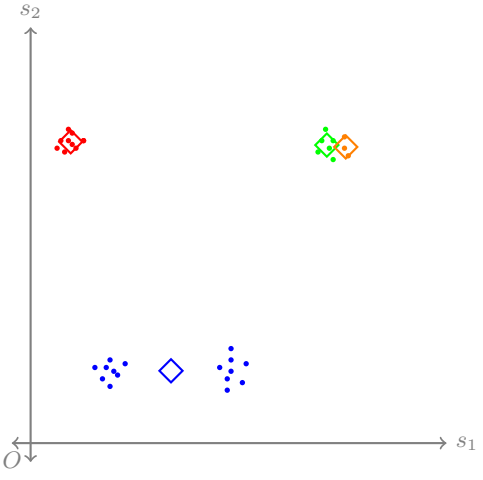
(c) Construction des clusters



(d) Mise à jour des positions des m_i

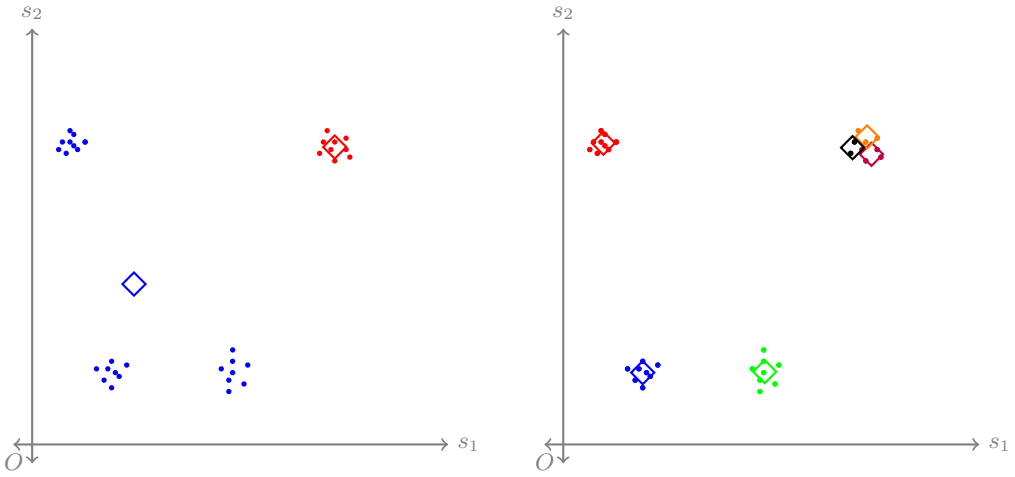


(e) Mise à jour des clusters



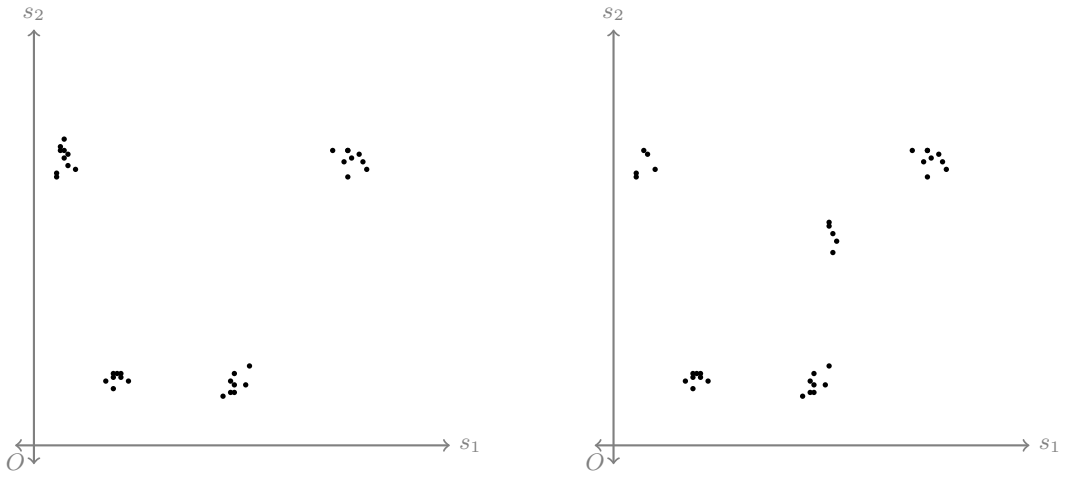
(f) Clusters finaux

FIGURE 5.9 – Exécution de l'algorithme κ-moyennes.



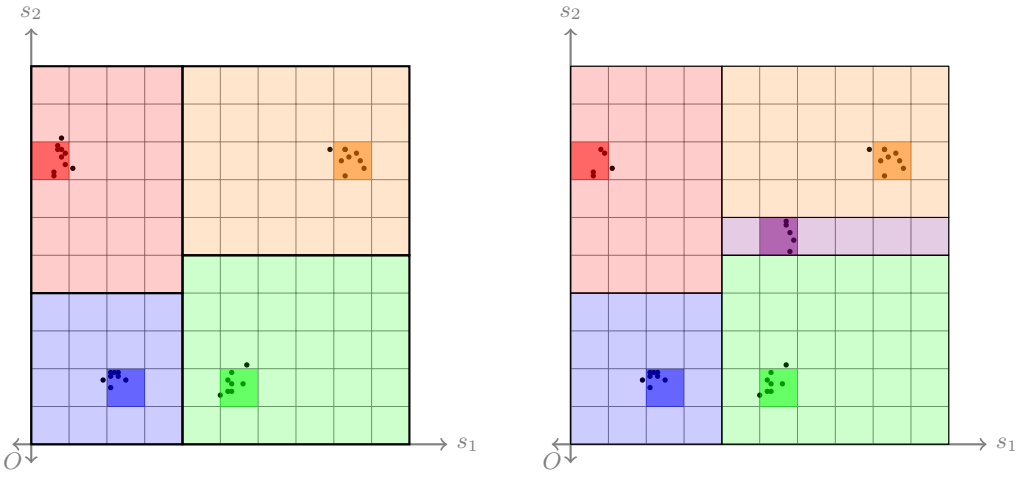
(a) $\kappa = 2$ (b) $\kappa = 6$

FIGURE 5.10 – Exécution de l'algorithme κ-moyennes pour différentes valeurs de κ.



(a) Valeur idéale de $\kappa = 4$ (b) Valeur idéale de $\kappa = 5$

FIGURE 5.11 – Évolution de la valeur idéale du paramètre κ à différents instants discrets.



(a) Valeur idéale de $\kappa = 4$ (b) Valeur idéale de $\kappa = 5$

FIGURE 5.12 – Évolution de FixMe en fonction de la position des entités.

à chaque instant qu'une entité appartient à la cellule régie par le référent le plus proche. Cette décomposition impose donc de connaître pour chaque référent le polyèdre qu'il gère ainsi que l'ensemble des entités appartenant à ce polyèdre. L'ajout d'un nouveau référent peut engendrer dans le pire des cas une redéfinition de chacun des polyèdres gérés par les référents existants ainsi qu'un changement de cluster pour un certain nombre d'entités. Cette approche peut donc s'avérer coûteuse lorsque le nombre de référents varie au cours du temps.

Comme nous l'avons vu, FixMe s'adapte dynamiquement au placement des entités dans l'espace des qualités et partitionnant l'espace E en cellules en fonction de la présence de seeds. À l'apparition d'un nouveau seed, la cellule C contenant ce nouveau seed est scindée en deux nouvelles cellules C_1, C_2 telles que $C = C_1 \cup C_2$. On a alors un partitionnement hiérarchique de l'espace des qualités qui diffère de celui effectué par l'algorithme κ-moyennes. Cependant, la structure des cellules de FixMe n'est présente que pour réduire les informations nécessaires aux entités pour se déplacer dans l'overlay. Par conséquent, le partitionnement de FixMe ne dégrade pas la qualité de la caractérisation effectuée par les algorithmes 12 et 13. De plus les autres cellules de FixMe ne sont pas impactées par l'apparition du nouveau seed rendant ainsi l'opération moins coûteuse que dans l'algorithme κ-moyennes.

5.7 Conclusion

Nous avons présenté dans ce chapitre FixMe une architecture auto-organisée dynamique et scalable permettant à des entités supervisées de trouver efficacement d'autres entités aux perceptions similaires. Cette architecture est décomposée en deux niveaux. Le premier gère l'espace des qualités et celui-ci est partitionné en cellules en fonction de

la forte présence d'entités dans différentes parties de cet espace. Le second niveau quant à lui organise les entités présentes dans une même région de l'espace au sein d'une DHT PeerCube. Les propriétés combinées de ces deux niveaux permettent ainsi d'obtenir une structure auto-organisante et tolérante au dynamisme des entités du système. La recherche de ces entités aux perceptions similaires permet ainsi d'appliquer le modèle et les algorithmes décrits dans le chapitre 3.

Bibliographie

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

[ALL+12] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : A Self-organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. Dans *Proceedings of the 16th International Conference On Principles Of Distributed Systems*, OPODIS, pages 1–12, décembre 2012.

[ALL+13] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : Détection Répartie de Défaillances Isolées. Dans *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, mai 2013.

[EJ01] D. Eastlake et P. Jones : US Secure Hash Algorithm 1 (SHA1). Rapport technique, IETF, 2001.

[HW79] J. A. Hartigan et M. A. Wong : Algorithm AS 136 : A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[LSW06] T. Locher, S. Schmid et R. Wattenhofer : eQuus : A Provably Robust and Locality-Aware Peer-to-Peer System. Dans *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, P2P, pages 3–11, 2006.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp et S. Shenker : A Scalable Content-addressable Network. Dans *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 161–172, 2001.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek et H. Balakrishnan : Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM Computer Communication Review*, 31(4):149–160, août 2001.

Chapitre 6

Étude locale de PeerCube

Nous avons présenté dans le chapitre précédent l'architecture de FixMe. Celle-ci permet de fournir aux entités supervisées la capacité de trouver efficacement d'autres entités effectuant des mesures de qualités similaires et percevant ainsi les mêmes variations de mesure, en vue d'appliquer les théorèmes décrits dans le chapitre 3.

FixMe partitionne l'espace des qualités E en buckets et cellules. Les buckets contenant plus de γ entités sont appelés *seeds*. Les entités des seeds s'organisent au sein d'une DHT PeerCube et gèrent les entités présentes dans la cellule. Cette DHT est nécessaire à notre approche pour quatre raisons.

Tout d'abord elle permet de gérer la population au sein de la cellule. De plus, elle est résistante au churn, c'est-à-dire à la propension des entités à entrer et sortir de celle-ci. Ensuite, elle permet de stocker de l'information qui est utilisée pour caractériser le type de faute induisant une défaillance perçue par des entités supervisées. Enfin, cette DHT est résistante aux comportements byzantins, c'est-à-dire que PeerCube assure la connectivité de sa structure ainsi que l'exécution des recherches dans la DHT malgré la présence d'entités ayant un comportement arbitraire par rapport au protocole PeerCube.

Dans ce chapitre, nous présentons une évaluation de PeerCube en présence d'un adversaire visant à mettre en défaut le protocole de PeerCube et ainsi remettre en cause la possibilité d'effectuer des recherches, lectures et écritures dans cette DHT. Après une présentation plus en profondeur de PeerCube, nous décrivons le modèle de l'adversaire employé ainsi que la modélisation de sa stratégie. Nous étudions ensuite l'impact de cette stratégie sur les clusters de PeerCube.

6.1 Fonctionnement de PeerCube

On considère un ensemble de n entités munies d'une identité unique issue d'une fonction de hachage $\mathfrak{h}$. Ces entités s'organisent dans FixMe comme l'illustre la figure 6.1. Considérons une cellule de FixMe, et supposons qu'elle est peuplée de $h \leq n$ entités à l'instant k. Ces h entités s'organisent alors au sein d'une instance PeerCube.

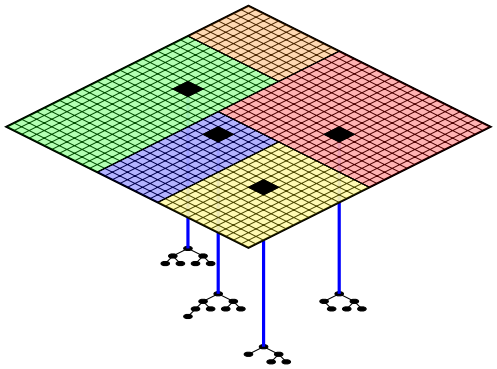


FIGURE 6.1 – Les deux niveaux de FixMe : l'espace des qualités partitionné en cellules selon la présence de seed et les DHT gérant la population de chaque seed.

6.1.1 Clusters

Dans PeerCube, les entités sont regroupés en clusters identifiés par un label. Chaque cluster contient un ensemble d'entités dont la taille est comprise entre γ et $\Gamma = \log h$. Afin de limiter les coûts de maintenance de cette structure seul un sous-ensemble des entités du cluster, appelé *core*, maintient les informations nécessaires à la gestion des entités du cluster. Cet ensemble est de taille fixe γ et forme une clique. Les entités du cluster n'appartenant pas au core forment le *spare*. Ces conditions sont décrites dans la propriété 3.

Propriété 3 (Structure) *Soit $\mathcal{C}$ un cluster de PeerCube, celui ci est composé de deux sous-ensembles C et S tels que :*
- $\mathcal{C} = C \cup S$
- $|C| = \gamma$
- $\gamma < |\mathcal{C}| < \Gamma$.

Exemple 22 *La figure 6.2 illustre la décomposition d'un cluster en core et spare. Le cluster est composé de* 12 *entités. Le core est constitué des $\gamma = 7$ entités bleues formant une clique au centre et le spare de* 5 *entités rouges sur la couronne extérieure.*

D'autre part, les entités sont regroupées au sein d'un cluster en fonction de leur identifiant. L'identifiant d'une entité est issu d'une la fonction de hachage $\mathfrak{h}$. Une fonction de hachage est une fonction qui projette une donnée fournie en entrée dans un espace de taille réduite. Par exemple, la fonction SHA-1 [EJ01] projette une donnée quelconque dans $[\![0, 1]\!]^{160}$. Les identifiants des entités sont représentés par une chaîne binaire.

Chaque cluster est identifié de manière unique par un label. Ce label vérifie les deux propriétés 4 et 5 suivantes.

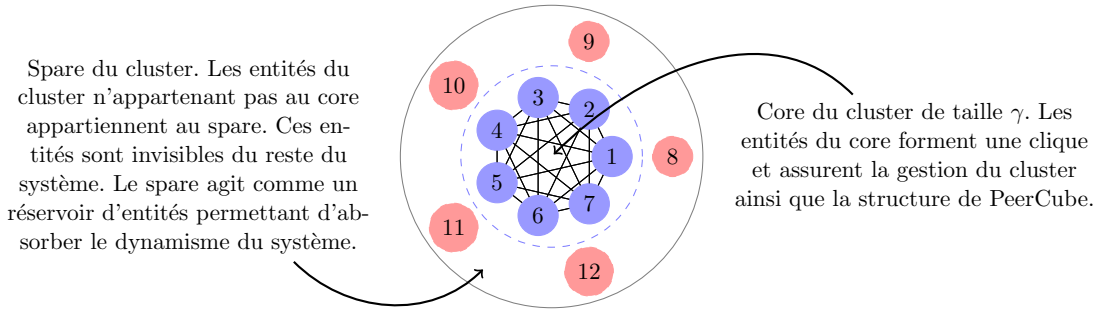


FIGURE 6.2 – Structure d'un cluster de 12 entités avec $\gamma = 7$.

Propriété 4 (Appartenance) *Soit $\mathcal{C}$ un cluster de PeerCube, pour toute entité p de $\mathcal{C}$, le label de $\mathcal{C}$ préfixe l'identifiant de p.*

Propriété 5 (Non-inclusion) *Soit $\mathcal{C}$ un cluster de PeerCube, pour tout cluster $\mathcal{C}'$ de PeerCube, le label de $\mathcal{C}' \neq \mathcal{C}$ ne préfixe pas celui de $\mathcal{C}$.*

La propriété 4 assure que chaque entité du système appartient à un unique cluster tandis que la propriété 5 assure que les clusters forment une partition de l'ensemble des entités présentes dans le système.

Le label d'un cluster est défini par le plus court préfixe d'identifiant commun partagé par les entités présentes dans le cluster et tel que la propriété 5 de non-inclusion soit satisfaite.

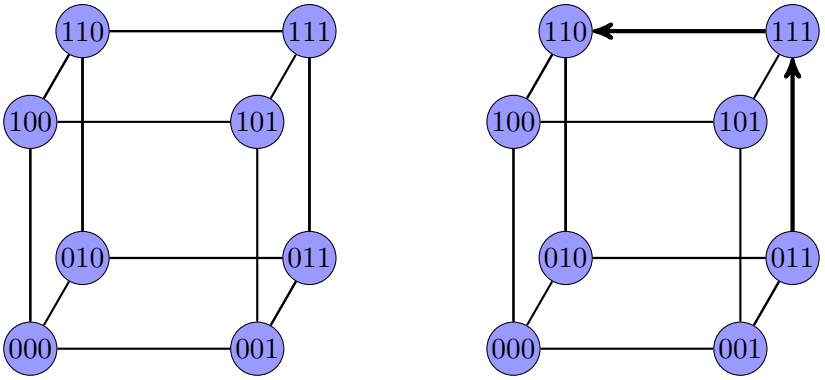
Exemple 23 *Considérons une instance PeerCube et un cluster labellisé "011" dans celle-ci. Dans ce cas, toutes les entités présentes dans ce cluster ont un identifiant débutant par "011". De plus, toutes les entités ayant un identifiant débutant par "011" appartiennent à ce cluster. Enfin, aucun cluster de cette instance de PeerCube ne peut avoir pour label "0", "01" ou tout autre label commençant par "011".*

6.1.2 Opérations de PeerCube

Le cluster constitue l'entité élémentaire de PeerCube. Les clusters se créent et disparaissent en fonction de la population présente dans PeerCube. Ces clusters s'auto-organisent en fonction du label de chacun.

Topologie Le label d'un cluster détermine la position de ce cluster dans la topologie globale. La topologie de PeerCube est basée sur un hypercube. Deux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$ de labels respectifs L_1 , L_2 sont voisins dans cet hypercube si la distance de Hamming entre leurs labels respectifs vaut 1. Cette distance compte le nombre de bits pour lesquels L_1 et L_2 diffèrent. La figure 6.3(a) représente une instance de PeerCube composée de 8 clusters. Les labels de ces clusters sont codés sur 3 bits de "000" à "111".

Nous décrivons à présent les opérations régissant le comportement des clusters.



(a) Structure de PeerCube. (b) Opération `lookup`.

FIGURE 6.3 – Structure et recherche dans PeerCube.

Opération `lookup` La procédure classique de `lookup` sur un hypercube allant d'un sommet v_1 à un sommet v_2 consiste à modifier successivement les bits du label v_1 qui diffèrent de celui de v_2 pour obtenir finalement le label de v_2.

Exemple 24 *La figure 6.3(b) illustre le déroulement de l'opération* `lookup` *sur la topologie précédente. Considérons qu'une entité du cluster labellisé "011" souhaite effectuer une opération* `lookup` *sur un identifiant préfixé par "110". La distance de Hamming entre ces deux labels vaut* 2*, les clusters correspondants ne sont donc pas voisins dans PeerCube.*

Les entités du core du cluster labellisé "011" modifient le premier bit du label du cluster qui diffère du label cible ("110"). On obtient alors "111". Une requête de `lookup` *est envoyée au cluster labellisé "111". Le core de ce cluster modifie le premier bit de "111" qui diffère du label cible et obtient "110". Les entités du core font suivre la requête de* `lookup` *au core du cluster "110". La requête est arrivée à destination.*

Cette opération `lookup` est la version la plus simple permettant de trouver un cluster dans la topologie PeerCube. Une seconde version sera présentée et évaluée dans le chapitre 7.

Opération `join` Lorsqu'une entité p entre dans le système, celle-ci doit trouver le cluster $\mathcal{C}$ auquel elle appartient. Elle contacte donc une entité q déjà présente dans PeerCube. Cette entité effectue alors une opération `lookup` afin de déterminer le cluster dont le label préfixe l'identifiant de p. L'entité p est alors insérée dans le spare de $\mathcal{C}$. La figure 6.4 illustre l'état d'un cluster avant et après une opération `join`. Le core reste inchangé tandis que le spare comporte une entité supplémentaire.

Opération `split` Suite à l'ajout d'une entité dans le cluster $\mathcal{C}$ labellisé L, la taille de ce cluster peut atteindre Γ. Dans ce cas, le cluster est scindé en deux nouveaux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$ labellisés respectivement $L0$ et $L1$.

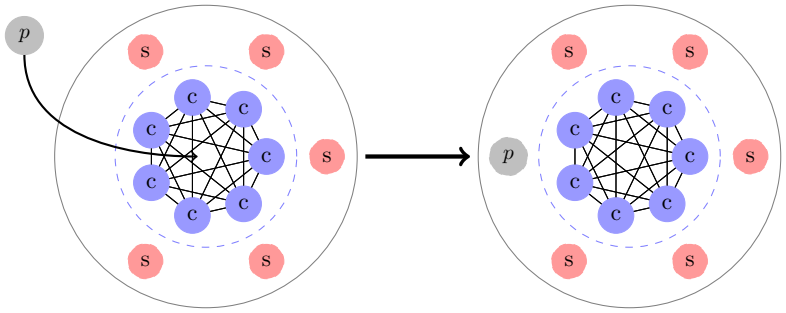


FIGURE 6.4 – Ajout d'une entité dans un cluster.

Le label $L0$ est obtenu en concaténant le label initial L avec "0". Le label $L1$ s'obtient en concaténant le label inital L avec "1". La taille du plus petit préfixe commun est augmentée afin de respecter la condition de population. Les entités de $\mathcal{C}$ sont alors insérées dans $\mathcal{C}_1$ ou $\mathcal{C}_2$ suivant leur identifiants. Chacun des deux nouveaux clusters gère alors la moitié de l'espace des identifiants précédemment géré par $\mathcal{C}$.

Cependant, il peut arriver qu'un des deux clusters $\mathcal{C}_1$, $\mathcal{C}_2$ ne vérifie pas la condition de population. En effet, les identifiants des entités sont issus d'une fonction de hachage. Par conséquent, il peut arriver que Γ entités aient des identifiants préfixés par L, tandis que moins de γ d'entre elles aient un identifiant préfixé par $L0$ (ou $L1$). Dans ce cas, le cluster $\mathcal{C}$ n'est pas scindé en deux nouveaux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$. La taille de $\mathcal{C}$ dépasse temporairement Γ. Dès que chaque nouveau cluster vérifie la condition de population, le cluster $\mathcal{C}$ est scindé. Cette situation se produit très rarement, par conséquent, nous l'ignorerons dans notre modélisation du protocole.

Exemple 25 *La figure 6.5 illustre le déroulement de cette opération. Considérons la figure 6.5 et supposons* $\gamma = 7, \Gamma = 23$. *Le cluster* $\mathcal{C}$ *représenté est initialement composé de* $|\mathcal{C}| = 22$ *entités. L'entité p rejoint dans le cluster, celle-ci est insérée dans le spare. Le cluster comporte alors* $|\mathcal{C}| = 23$ *entités : il est sur-peuplé. Le cluster doit donc être scindé en deux nouveaux clusters* $\mathcal{C}_1$ *et* $\mathcal{C}_2$. *La séparation des entités dans chacun des nouveaux clusters est faite suivant l'identifiant de chaque entités. Par conséquent, les deux nouveaux clusters n'ont pas nécessairement une taille identique. Les deux nouveaux clusters sont de tailles respectives* $|\mathcal{C}_1| = 13$ *et* $|\mathcal{C}_2| = 10$.

Opération `leave` Lorsqu'une entité p quitte le système, deux cas se présentent. Tout d'abord, si p appartient au spare du cluster, l'entité est simplement supprimée du spare. De cette manière, le départ de p est transparent du point de vue du reste du système. Le spare joue alors le rôle d'un réservoir d'entités permettant la diminution des messages de mise à jour nécessaire pour le maintien de la cohérence de la structure et absorbant ainsi un partie du dynamisme des entités.

À l'inverse, si p appartient au core du cluster, l'entité p est supprimée du core. Dans ce cas, le core est sous-peuplé, il faut donc renouveler sa composition. On désigne par $P_\phi, 1 \leq \phi \leq \gamma$, le protocole de PeerCube décrivant ce renouvellement.

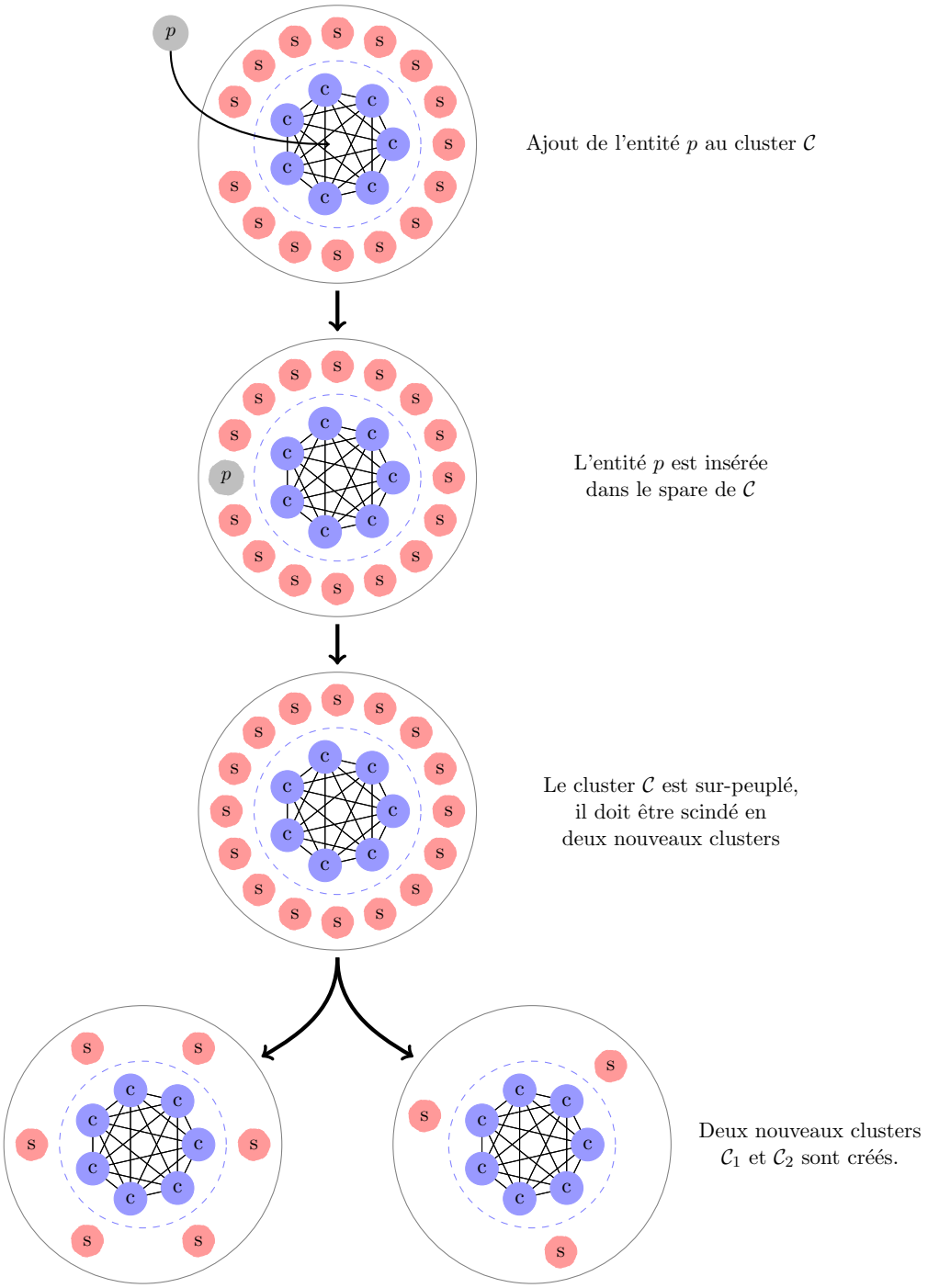


FIGURE 6.5 – Ajout d'une entité dans un cluster et création de deux nouveaux clusters. Ici, $\gamma = 7, \Gamma = 23$

La procédure de renouvellement consiste à sélectionner $\phi - 1$ entités du core et à la placer temporairement dans le spare. Ensuite, ϕ entités présentes dans le spare sont sélectionnées pour faire partie du core. Le core comporte de nouveau γ entités. L'impact du paramètre ϕ est étudié dans ce chapitre.

Exemple 26 *La figure 6.6 illustre le déroulement de l'opération de renouvellement du core du cluster dans le cas où $\gamma = 7$ et $\phi = 4$. Initialement, le cluster $\mathcal{C}$ est composé de 13 entités avec $\gamma = 7$. Une entité du core quitte le cluster, celui-ci ne contient plus γ entités, il faut donc renouveler sa composition. La procédure de renouvellement sélectionne aléatoirement $\phi - 1$ entités du core pour les insérer dans le spare. La procédure sélectionne ensuite ϕ entités présentes dans le spare pour les placer dans le core. À la fin de la procédure, le core est de nouveau constitué de γ entités. Deux entités initialement présentes dans le core sont à présent situées dans le spare du cluster, tandis que trois entités présentes dans le spare initialement se retrouvent à présent dans le core de celui-ci.*

Opération `merge` Suite au départ d'une entité du cluster $\mathcal{C}_1$ labellisé $L0$, la taille du cluster peut atteindre γ. Dans ce cas, le cluster est fusionné avec le cluster $\mathcal{C}_2$ labellisé $L1$. Le préfixe du nouveau cluster est obtenu en calculant le préfixe commun L aux deux labels $L0$ e $L1$. Le nouveau cluster $\mathcal{C}$ est alors labellisé L et son core comporte les entités du core de $\mathcal{C}_2$. Les entités du spare de $\mathcal{C}_2$ et du core de $\mathcal{C}_1$ forment le spare de $\mathcal{C}$.

Il peut arriver que le cluster labellisé $L1$ n'existe pas. En effet, ce cluster a pu être scindé en deux nouveaux clusters. Dans ce cas, le cluster $\mathcal{C}_1$ est fusionné avec l'ensemble des clusters dont les labels sont préfixés par $L1$.

Exemple 27 *La figure 6.7 illustre le déroulement de cette opération. Considérons le cluster $\mathcal{C}_1$: l'entité présente dans le spare de $\mathcal{C}_1$ quitte le cluster. On a alors $|\mathcal{C}_1| = 7 = \gamma$, le cluster doit fusionner avec le cluster $\mathcal{C}_2$ représenté à droite de $\mathcal{C}_1$. Un nouveau cluster $\mathcal{C}$ est créé dont le core est composé des entités du core de $\mathcal{C}_2$. Le spare est composé des entités restantes de $\mathcal{C}_1$ et du spare de $\mathcal{C}_2$.*

6.2 Adversaire

Comme nous l'avons dit la présence de fautes est inévitable. En particulier, il peut arriver que des entités supervisées ne suivent plus le protocole défini. On parle alors de faute byzantine. Nous étudions dans ce chapitre la résilience de PeerCube en présence de telles fautes.

Nous considérons par la suite un adversaire cherchant à manipuler le système. Celui-ci contrôle totalement un sous-ensemble des entités supervisées. Les entités contrôlées par l'adversaire sont dites *malveillantes*. À l'inverse, une entité exécutant le protocole PeerCube est appelée entité *honnête*. Une entité honnête ne peut pas distinguer une autre entité honnête d'une entité malveillante. Le cas où les entités malveillantes

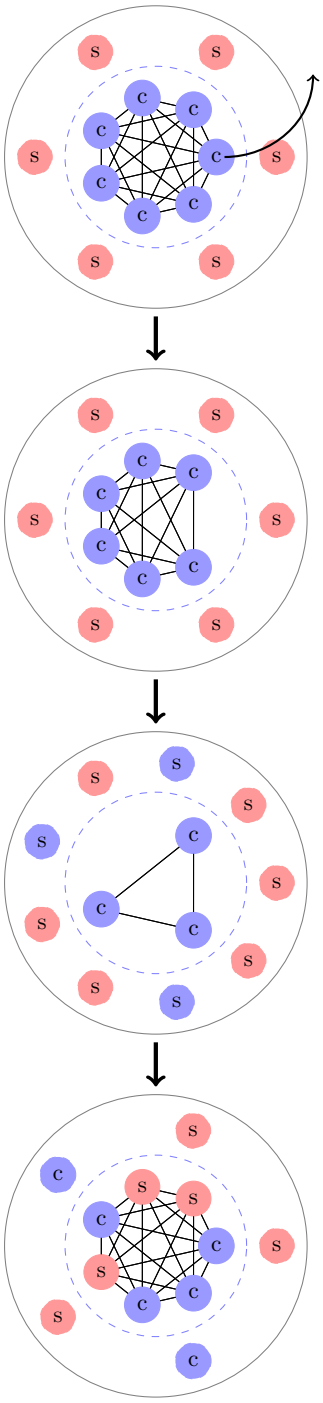


FIGURE 6.6 – Renouvellement de la composition du core suite au départ d'une entité de celui-ci. Ici, $\gamma = 7, \phi = 4$.

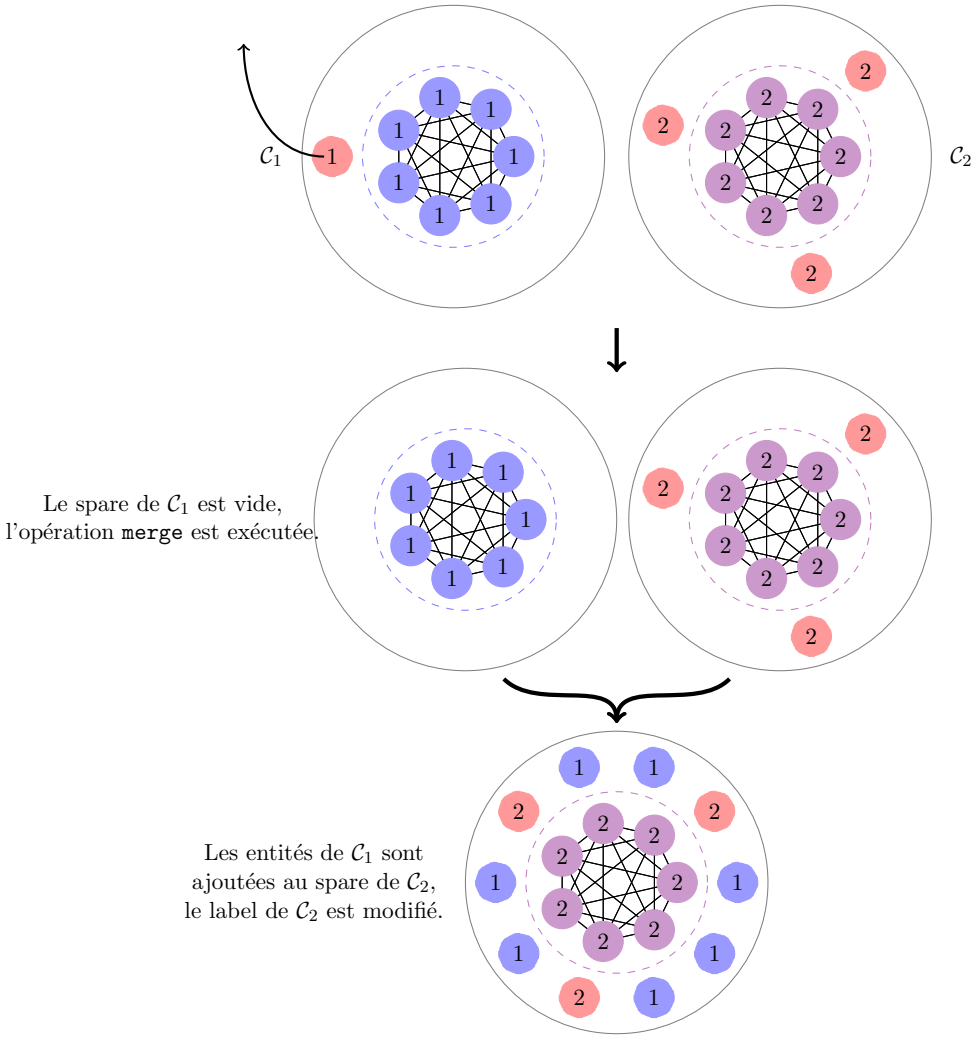


FIGURE 6.7 – Suppression d'une entité dans un cluster et fusion des clusters.

Notation	Signification
n	Nombre d'entités présentes dans le système
h	Nombre d'entités présentes dans la cellule FixMe courante (*i.e.*, dans l'instance PeerCube considérée)
$\mathfrak{h}$	Fonction de hachage utilisée pour déterminer les identifiants dans PeerCube
γ	Taille minimale d'un cluster
Γ	Taille maximale d'un cluster
$\mathcal{C} = \{C, S\}$	Cluster composé d'un core C de taille γ et d'un spare S
$\Delta = \Gamma - \gamma$	Taille maximale du spare
P_ϕ	Protocole de renouvellement du core dans lequel suite au départ d'une entité de celui-ci
c	Seuil de pollution d'un cluster

TABLE 6.1 – Liste des notations

agissent en collusion représente le pire cas permettant d'étudier la tolérance aux fautes du système.

Nous considérons que cet adversaire contrôle une proportion μ ($0 \leq \mu < 1$) des n entités du système. L'adversaire est donc susceptible de contrôler à un instant donné un nombre $\mu n > \mu h$ d'entités dans l'instance PeerCube considérée.

Nous faisons l'hypothèse ici de l'utilisation d'outils de cryptographie empêchant une entité malveillante de se forger diverses identités. De plus, les entités malveillantes n'ont aucun contrôle sur les messages échangés entre entités honnêtes. En revanche, ces entités ont un contrôle total sur les messages qu'elles émettent ou reçoivent.

6.2.1 Identifiants

Il a été montré dans [AS07] que les overlays structurés ne peuvent résister à un adversaire que s'ils assurent les deux propriétés suivantes :

- **Uniformité :** les identifiants des entités dans l'espace d'adressage sont distribués uniformément
- **Dynamicité :** aucune entité ne peut rester indéfiniment dans une même région de l'espace d'adressage.

Afin d'implémenter ces deux conditions dans PeerCube, on introduit une durée de vie pour chaque identifiant. Ces identifiants vérifient les quatre propriété suivantes :

1. les identifiants ont une durée de vie limitée,
2. les identifiants ne peuvent pas être forgés,
3. les identifiants sont imprédictibles,
4. les identifiants sont vérifiables par une autre entité.

Une implémentation possible de ces identifiants consiste à attribuer par une autorité de certification un certificat X.509 [HFPS99] à chaque entité p du système. Ainsi, les

identifiants ne peuvent être forgés par une entité du système. L'identifiant initial id_p^0 est ensuite obtenu en appliquant la fonction de hachage $\mathfrak{h}$ sur l'un des champs de ce certificat. L'utilisation de la fonction de hachage assure ainsi l'uniformité des identifiants sur l'espace d'adressage et rendent les identifiants imprédictibles.

Nous souhaitons forcer les entités du système (honnêtes comme malveillantes) à se replacer régulièrement dans le système. De cette manière, aucune entité ne peut rester indéfiniment dans une même région de l'espace d'adressage. L'autorité de certification ajoute au certificat un date de création t_0, correspondant à la date d'obtention du certificat. Le certificat est inaltérable par une entité malveillante.

On fixe de plus une durée de vie ℓ aux identifiants. Ainsi, à l'instant t, une entité p utilise son i-ème identifiant $id_p^i = \mathfrak{h}(id_p^0 \times i)$, avec $i = \lceil (t - t_0)/\ell \rceil$. Par conséquent, la validité de cet identifiant expire à l'instant $t = t_0 + i\ell$. L'intervalle de temps $[t_0 + (i - 1)\ell, t_0 + i\ell]$ représente la fenêtre de validité du i-ème identifiant.

Toute entité du système peut lire le certificat d'une autre entité. Ainsi, une entité p peut confirmer la validité de l'identifiant d'une autre entité q en calculant son i-ème identifiant id_q^i. On a alors la propriété suivante :

Propriété 6 (Durée de vie des identifiants) *Soient $\mathcal{C}$ un cluster de PeerCube labellisé L et p une entité du système. L'entité p appartient au cluster $\mathcal{C}$ à l'instant t si et seulement si l'identifiant de p est valide à l'instant t et si L préfixe cet identifiant.*

Si une entité $q \in \mathcal{C}$ arbore un identifiant invalide ou périmé et qu'une autre entité p du cluster $\mathcal{C}$ en prend conscience en vérifiant la validité de cet identifiant, p rompt le lien de communication avec q. Ceci a pour effet de forcer l'entité q à recalculer un nouvel identifiant valide et donc à se replacer dans le système conformément à ce nouvel identifiant.

Ce calcul d'identifiants repose sur les horloges locales de chaque entité du système. Ces horloges n'ont qu'une synchronisation légère (par exemple NTP [MMBK10]). Par conséquent, une entité $p \in \mathcal{C}$ honnête peut utiliser son i-ème identifiant id_p^i tandis que les autres entités du cluster $\mathcal{C}$ la jugent invalide.

Afin de prendre en compte cette faible synchronisation des horloges, on autorise que deux identifiants consécutifs id_p^i et id_p^{i+1} soient considérés comme valides au même instant t. Plus précisément, étant donné w la déviation maximale d'horloge entre deux entités correctes, avec $w << \ell$, il existe un intervalle de temps de longueur w tel que les deux identifiants id_p^i et id_p^{i+1} de $p \in \mathcal{C}$ soient considérés comme valides par les autres entités $q \in \mathcal{C}$. On a alors :

$$i = \lceil (t - w/2 - t_0)/\ell \rceil \text{ et } i + 1 = \lceil (t + w/2 - t_0)/\ell \rceil.$$

La figure 6.8 illustre cette marge de tolérance. L'axe du temps est découpé en intervalles de longueur ℓ. Le i-ème intervalle correspond à la durée de validité de l'identifiant id_p^i de l'entité p. Les intervalles grisés représentent la marge de tolérance de validité additionnelle pour les identifiants. Cette tolérance intervient dans un intervalle de temps de longueur $w/2$ avant le début effectif de la validité de l'indetifiant id_p^i et après la péremption de celui-ci.

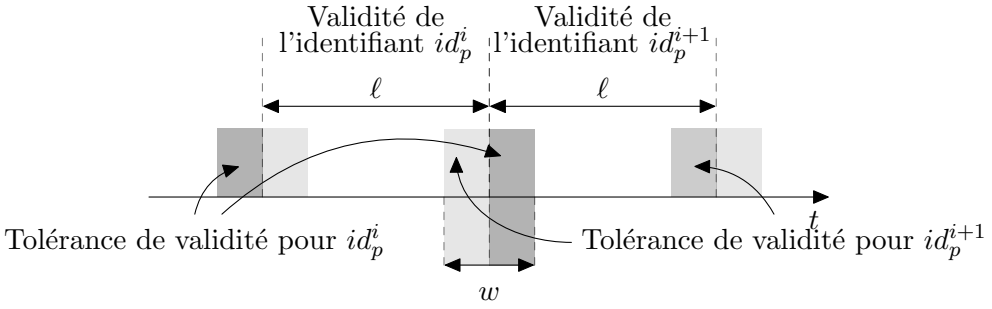


FIGURE 6.8 – Intervalles de validité des identifiants successifs de l'entité p.

Forcer les entités du système à se replacer régulièrement au sein de celui-ci induit une hyper-activité du système appelée churn induit. Nous étudions dans ce chapitre l'impact du churn induit sur le comportement des clusters de PeerCube.

6.2.2 Stratégie de l'adversaire

La gestion des entrées/sorties au niveau du cluster peut-être assurée au moyen d'algorithmes de consensus tolérants aux byzantins, permettant ainsi d'assurer la mise à jour des tables de routage et de la composition des entités du cluster, même en présence d'entités au comportement arbitraire. Ce type d'algorithme nécessite en général $\mathcal{O}(m^3)$ messages afin d'effectuer un consensus entre m entités. Dans notre cas, seules les entités du core sont concernées par l'utilisation de ce type d'algorithme. Le core est de taille constante $\gamma \ll h < n$, l'utilisation de ce type d'algorithme n'est donc pas trop coûteuse dans notre cas.

Malgré l'utilisation d'algorithmes de consensus tolérants aux byzantins, la présence d'entités malveillantes peut amener les clusters à avoir un comportement erroné. En effet, dès qu'un quorum d'entités malveillantes est réuni au sein du core d'un cluster, celui-ci peut imposer les décisions prises lors des algorithmes de consensus exécutés pour la mise à jour des tables de routage ou de la vue du cluster.

Dans le cas du modèle de défaillances byzantines authentifié, la présence de $\lfloor(\gamma - 1)/2\rfloor + 1$ entités malveillantes au sein d'une population de γ entités est suffisante pour empêcher les entités honnêtes d'imposer leur décision. Lorsque ce quorum est atteint, on dit que le cluster est *pollué*, sinon le cluster est dit *sain*. Nous notons $c = \lfloor(\gamma-1)/2\rfloor$ le seuil de pollution.

Afin de maximiser la probabilité de pollution des clusters, l'adversaire ne retire jamais ses entités malveillantes du système. Ainsi, elles restent à une même place le plus longtemps possible augmentant ainsi la proportion locale d'entités malveillantes et donc la possibilité de corrompre un cluster. Cependant, il peut arriver que le départ d'une entité malveillante du core engendre un renouvellement de celui-ci permettant ainsi de corrompre le cluster.

Afin de décider s'il doit forcer le départ d'une entité malveillante, l'adversaire estime la probabilité d'augmenter la représentation des entités malveillantes au sein du core.

Étant donné un cluster $\mathcal{C}$ dont le core contient x entités malveillantes et le spare y, l'adversaire calcule la probabilité d'atteindre tous les états où la proportion d'entités malveillantes au sein du core est plus importante que dans l'état actuel.

Si cette probabilité est supérieure à un paramètre η fixé de l'adversaire, alors celui-ci déclenche le départ d'une entité malveillante. Le paramètre η représente la marge d'erreur que l'adversaire tolère à défaut de pouvoir avoir un contrôle total sur le cluster. Ce paramètre peut être assimilé au risque statistique fixé par l'adversaire. Cette décision est décrite formellement par la règle suivante :

Règle 1 (Stratégie des départs) *Soit $\mathcal{C} = \{C, S\}$ un cluster dont le core C contient $0 < x \leq c$ entités malveillantes et le spare S contient $y > 1$ entités malveillantes. Étant donné $0 < \eta < 1$, l'adversaire décide de retirer une entité malveillante du core si :*

$$\Pr\{|\{\textit{entités malveillantes dans } C \textit{ après renouvellement}\}| > x\} > 1 - \eta. \tag{6.1}$$

Dans le cas où $\phi = 1$, le protocole P_ϕ choisit une unique entité dans le spare pour compenser le départ d'une entité du core. Par conséquent, l'adversaire peut au mieux conserver le même nombre d'entités malveillantes au sein du core. Dans ce cas, cette règle n'est jamais appliquée. Dans le cas où $\phi > 1$, l'entité malveillante ayant l'identifiant le plus proche de son instant d'expiration est choisie pour quitter le core et ainsi déclencher le renouvellement de celui-ci.

Une fois le cluster pollué, l'adversaire souhaite le maintenir le plus longtemps possible dans cet état. Ainsi, dès qu'un départ du core se produit, il remplace l'entité ayant quitté le cluster par une entité malveillante. Cependant, il peut arriver qu'il n'y ait plus suffisamment d'entités malveillantes présentes dans le cluster. Dans ce cas, l'adversaire n'a d'autres choix que de perdre le contrôle de celui-ci.

Dans le cas où un cluster devient sous-peuplé ($|\mathcal{C}| = \gamma$) ou surpeuplé ($|\mathcal{C}| = \Gamma$), un changement topologique s'opère dans PeerCube. Comme nous l'avons vu précédemment, dans le cas où un cluster $\mathcal{C}_1$ devient sous-peuplé, celui-ci doit fusionner avec un autre cluster $\mathcal{C}_2$ pour former un seul cluster $\mathcal{C}$. Dans ce cas, les entités du core de $\mathcal{C}_1$ sont placées dans le spare de $\mathcal{C}$ avec les entités du spare de $\mathcal{C}_2$ tandis que les entités du core de $\mathcal{C}_2$ forment le core du nouveau cluster $\mathcal{C}$. Ainsi, dans le cas où $\mathcal{C}_1$ est pollué, l'adversaire perd tout contrôle sur le cluster. Il cherche donc à tout prix à éviter cette situation. Pour cette raison, les entités malveillantes ayant une identité valide au sens de la propriété 6 ne quittent jamais le cluster.

De manière similaire, lorsqu'un cluster devient surpeuplé ($|\mathcal{C}| = \Gamma$) celui-ci est scindé en deux nouveaux clusters. Les entités du cluster originel se répartissent dans ces deux nouveaux clusters selon leurs identifiants. Ainsi, si le cluster $\mathcal{C}$ est pollué, l'adversaire n'a aucune assurance que les deux nouveaux clusters soient encore pollués. De plus, l'espace d'identifiants géré par $\mathcal{C}$ est le même que celui géré par les deux nouveaux clusters. L'adversaire n'a donc aucun intérêt à laisser un cluster pollué se scinder en deux nouveaux clusters. Afin d'éviter cette situation, l'adversaire ignore les requêtes `join` provenant d'entités honnêtes si le spare contient suffisamment d'entités. La règle suivante décrit le comportement de l'adversaire vis-à-vis des changements topologiques.

Règle 2 (Stratégie des arrivées) *On considère un cluster pollué $\mathcal{C}$ dont le spare contient $0 < s < \Delta$ entités avec $\Delta = \Gamma - \gamma$. Une requête de* **`join`** *envoyée par une entité j au cluster $\mathcal{C}$ est ignorée si (j est honnête et $s > 1$) ou si ($s = \Delta - 1$), Δ représentant la taille maximale du spare.*

Ces deux règles nous permettent à présent de modéliser l'évolution de la composition d'un cluster en présence de l'adversaire.

6.3 Modélisation d'un cluster

Les clusters sont composés d'une ensemble d'entités de taille fixe γ appelé core et d'un ensemble d'entités de taille variable s appelé spare. La taille du cluster est comprise entre γ et Γ. On note $\Delta = \Gamma - \gamma$. La taille d'un cluster est donc totalement caractérisée par la taille de son spare $0 < s < \Delta$.

D'autre part, les clusters sont peuplés par deux types d'entités : certaines honnêtes et d'autres malveillantes. Par conséquent, si on note $0 \leq x \leq \gamma$ le nombre d'entités malveillantes dans le core et $0 \leq y \leq s$ le nombre d'entités malveillantes dans le spare, le triplet (s, x, y) décrit totalement l'état du cluster.

Exemple 28 *La figure 6.9 illustre la composition de deux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$ pour $\gamma = 7$ et $\Gamma = 14$. Les entités de la clique centrale forment le core du cluster et les entités de la couronne extérieure forment le spare. On a $|\mathcal{C}_1| = |\mathcal{C}_2| = 13, s = 6$. Les entités honnêtes sont représentées par le symbole ☺ sur fond bleu tandis que les entités malveillantes sont représentées par le symbole ☠ sur fond rouge.*

Le cluster $\mathcal{C}_1$ représenté sur la figure 6.9(a) comporte 3 entités malveillantes dans le core et 2 entités malveillantes dans le spare. On représente l'état du cluster $\mathcal{C}_1$ par le triplet $(6, 3, 2)$.

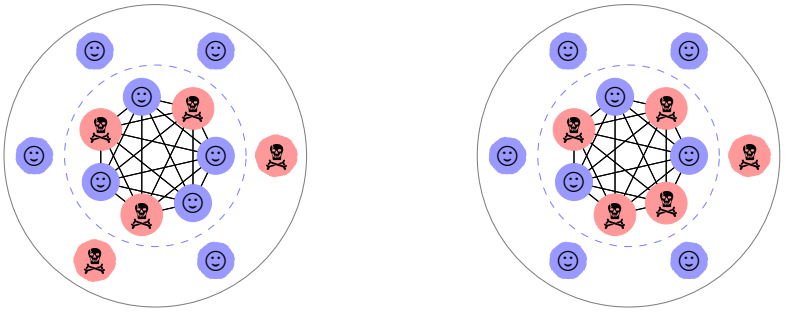
Le cluster $\mathcal{C}_2$ représenté sur la figure 6.9(b) comporte 4 entités malveillantes dans le core et 1 entités malveillantes dans le spare. On représente l'état du cluster $\mathcal{C}_2$ par le triplet $(6, 4, 1)$.

Le seuil de pollution c est défini par $c = \lfloor(\gamma - 1)/2\rfloor$, on a ici $c = \lfloor(7 - 1)/2\rfloor = 3$. Par conséquent, le cluster $\mathcal{C}_1$ est sain tandis que le cluster $\mathcal{C}_2$ est pollué.

L'évolution de la composition d'un cluster dépend des trois éléments suivants :
— les événements `join` et `leave` des entités du cluster
— le protocole P_ϕ employé pour le renouvellement du core
— la stratégie de l'adversaire que nous venons de décrire dans la section précédente.

On modélise l'effet des événements `join` et `leave` sur un cluster $\mathcal{C}$ par une chaîne de Markov homogène à temps discret notée $X^{(\phi)} = \{X_m^{(\phi)}, m \geq 0\}$. Celle-ci représente l'évolution de l'état du cluster $\mathcal{C}$, c'est à dire l'évolution de la taille du spare, du nombre d'entités malveillantes dans le core ainsi que du nombre d'entités malveillantes présentes dans le spare. L'espace d'états $\mathcal{X}$ de la chaîne $X^{(\phi)}$ est défini ainsi :

$$\mathcal{X} = \{(s, x, y) \mid 0 \leq s \leq \Delta,\ 0 \leq x \leq \gamma,\ 0 \leq y \leq s\}.$$



(a) $s = 6, x = 3, y = 2$ (b) $s = 6, x = 4, y = 1$

FIGURE 6.9 – Composition de deux clusters $\mathcal{C}_1, \mathcal{C}_2$. On a $|\mathcal{C}_1| = |\mathcal{C}_2| = 13, s = 6$.

Pour $m \geq 1$, l'événement $X_m^{(\phi)} = (s, x, y)$ désigne l'état du cluster $\mathcal{C}$ après la m-ème transition (*i.e.*, après le m-ème événement `join` ou `leave`) dans le cas du protocole P_ϕ. On a alors $|\mathcal{C}| = \gamma + s$ entités dans le cluster, dont x sont malveillantes et appartiennent au core et y sont malveillantes et appartiennent au spare.

On dit qu'un état est *pollué* si le core du cluster contient plus de $c = \lfloor (\gamma - 1)/2 \rfloor$ entités malveillantes. Dans le cas contraire, l'état est *sain*. L'ensemble des états sains S et l'ensemble des états pollués sont :

$$S = \{(s, x, y) \in \mathcal{X} \mid 0 < s < \Delta,\ 0 \leq x \leq c,\ 0 \leq y \leq s\},$$

$$P = \{(s, x, y) \in \mathcal{X} \mid 0 < s < \Delta,\ c + 1 \leq x \leq \gamma,\ 0 \leq y \leq s\}.$$

Le cluster alterne entre des états sains et des états pollués avant de disparaître du système. Cela arrive dans deux cas. Lorsque le cluster $\mathcal{C}$ devient sous peuplé ($|\mathcal{C}| = \gamma$), il doit alors fusionner avec un autre cluster (opération `merge` *i.e.* la taille du spare est nulle). De manière symétrique, lorsque le cluster $\mathcal{C}$ devient surpeuplé ($|\mathcal{C}| = \Gamma$), il doit être scindé en deux nouveaux clusters (opération `split` *i.e.*, le spare a atteint sa taille maximale Δ).

On modélise la disparition d'un cluster $\mathcal{C}$ par les états de $\mathcal{X}$ pour lesquels $s = 0$ ou $s = \Delta$. Ces états sont absorbants. On distingue de plus parmi les états où le spare a atteint sa taille maximale, les états où le cluster est pollué de ceux où il est sain.

On a les trois classes d'états absorbants suivantes :

$$A_S^\omega = \{(0, x, 0) \in \mathcal{X} \mid 0 \leq x \leq c\},$$

$$A_P^\omega = \{(0, x, 0) \in \mathcal{X} \mid c + 1 \leq x \leq \gamma\},$$

$$A_S^\sigma = \{(\Delta, x, y) \in \mathcal{X} \mid 0 \leq x \leq c, 0 \leq y \leq \Delta\}.$$

L'ensemble A_S^ω correspond aux états où le spare a une taille nulle et le cluster est sain, A_P^ω correspond aux états où le spare a une taille nulle et le cluster est pollué et enfin A_S^σ correspond aux états où le cluster est sain et le spare a atteint sa taille maximale Δ.

Exemple 29 *La figure 6.10 illustre la décomposition de l'espace d'états $\mathcal{X}$. Chaque état de $\mathcal{X}$ est représenté par un point de coordonnées (s, x, y). Les points bleus correspondent aux états $(0, x, 0)$, c'est à dire aux états où le cluster est sous-peuplé et doit fusionner avec un autre cluster. Les points violets correspondent aux états (Δ, x, y) c'est à dire aux états où le cluster est sur-peuplé et doit être scindé en deux nouveaux clusters. Les états $(0, x, 0)$ et (Δ, x, y) sont absorbants. Les états (s, x, y) représentés par un cercle correspondent aux états où $x \leq c$, c'est-à-dire aux états sains. Les autres états (représentés en diamant) correspondent aux états pollués. Enfin, tous les états rouges correspondent aux états transitoires.*

Du fait de la règle 2, l'adversaire empêche un cluster pollué de se scinder en deux nouveaux clusters. Par conséquent, les états pollués où le spare a atteint sa taille maximale Δ ne sont pas atteignables. On réduit donc l'espace d'états $\mathcal{X}$ à l'ensemble des états atteignables depuis les états de P ou S. On a $\mathcal{X} = S \cup P \cup A_S^\omega \cup A_S^\sigma \cup A_P^\omega$. La figure 6.11 illustre cette décomposition de l'espace d'états $\mathcal{X}$ dans le cas où $\gamma = 7$ et $\Delta = 10$.

Ces ensembles d'états ont pour cardinal :

$$|A_S^\omega| = \lfloor \frac{\gamma - 1}{2} \rfloor + 1 = c + 1$$

$$|A_P^\omega| = \gamma - \lfloor \frac{\gamma - 1}{2} \rfloor = \gamma - c$$

$$|A_S^\sigma| = (\Delta + 1) \times \big(\lfloor \frac{\gamma - 1}{2} \rfloor + 1 \big) = (\Delta + 1)(c + 1)$$

$$|S| = \big(\lfloor \frac{\gamma - 1}{2} \rfloor + 1 \big) \times \sum_{s=1}^{\Delta - 1} (s + 1) = \big(c + 1 \big) \frac{(\Delta + 2)(\Delta - 1)}{2}$$

$$|P| = \big(\gamma - \lfloor \frac{\gamma - 1}{2} \rfloor \big) \times \sum_{s=1}^{\Delta - 1} (s + 1) = \big(\gamma - c \big) \frac{(\Delta + 2)(\Delta - 1)}{2}$$

$$|\mathcal{X}| = (\gamma + 1) \Big(\frac{(\Delta + 2)(\Delta - 1)}{2} + 1 \Big) + (\Delta + 1) \big(c + 1 \big)$$

Le cluster alterne entre des états sains (états de S) et des états pollués (états de P) avant de disparaître du système, c'est-à-dire avant d'atteindre un état absorbant des classes A_S^ω, A_S^σ ou A_P^ω. La figure 6.12 illustre une vue agrégée des états de la chaîne de Markov $X^{(\phi)}$ ainsi que des transitions possibles entre ces classes d'états. Les états transitoires sains sont représentés par S, et les transitoires pollués par P. Les classes A_S^ω, A_P^ω et A_S^σ contiennent les états absorbants.

Le paramètre ϕ lié au nombre d'entités renouvelées suite à un départ du core n'intervient pas dans la décomposition de l'espace d'états. En revanche ce paramètre a une influence sur les transitions entre les états. On note $M^{(\phi)}$ la matrice de transition de la chaîne $X^{(\phi)}$. Celle-ci est partitionnée de la même manière que l'espace d'états $\mathcal{X} = S \cup P \cup A_S^\omega \cup A_S^\sigma \cup A_P^\omega$.

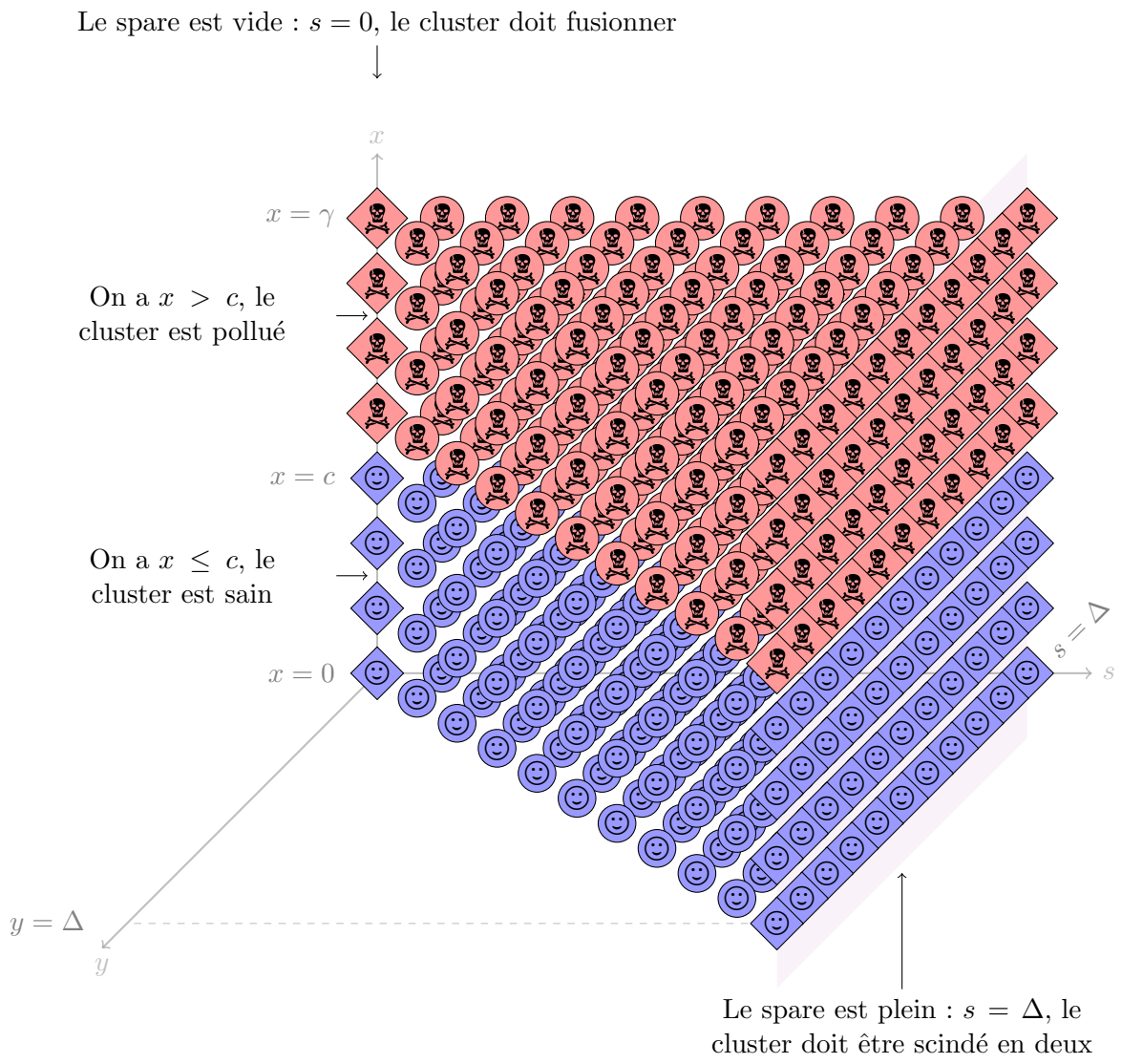


FIGURE 6.10 – Espace d'états $\mathcal{X}$ de la chaîne de Markov $X^{(\phi)}$. Dans le cas où $\gamma = 7$ et $\Delta = 10$, l'espace d'états contient 484 états.

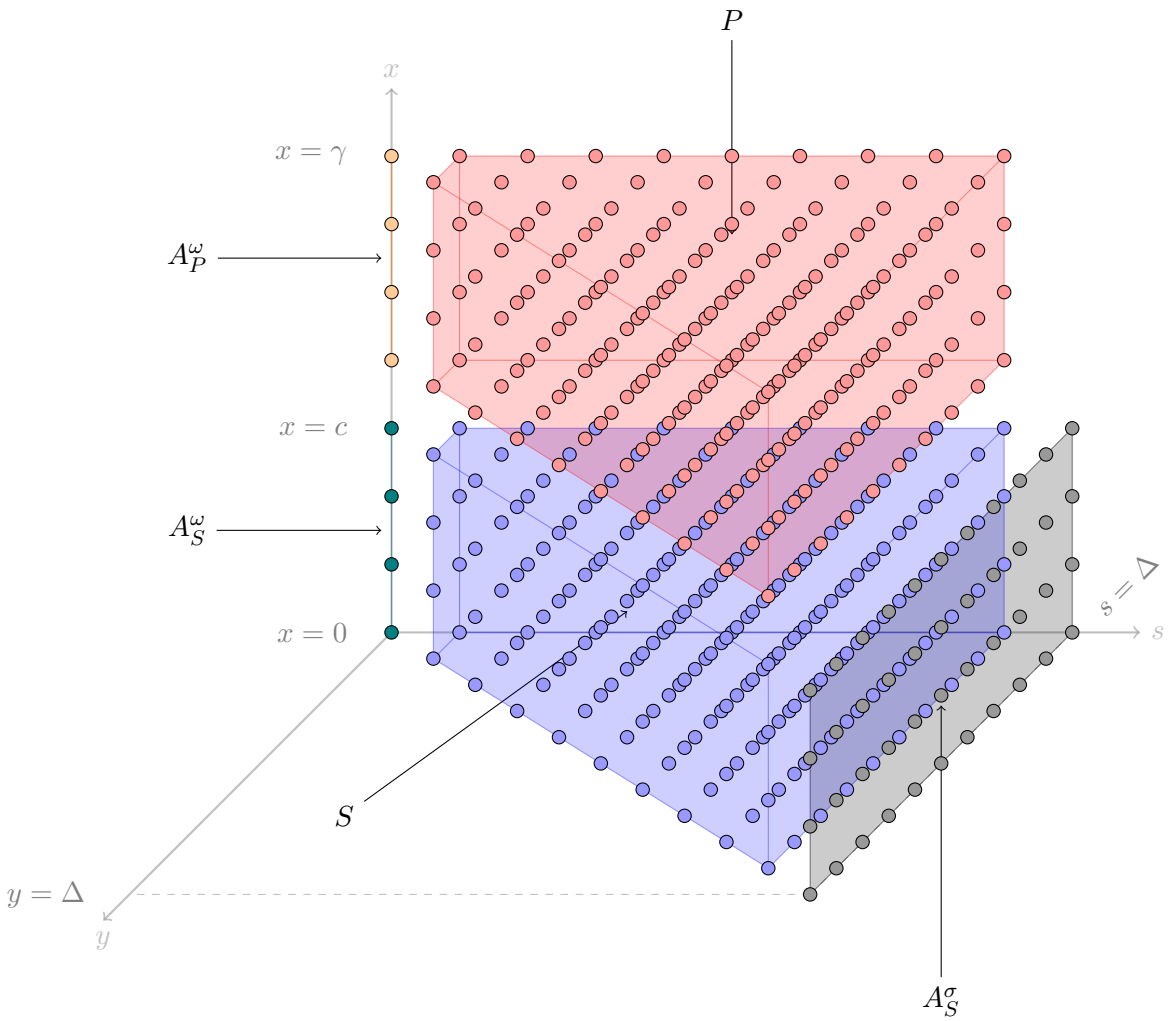


FIGURE 6.11 – Décomposition de l'espace d'états $\mathcal{X}$ de la chaîne de Markov $X^{(\phi)}$. Pour $\gamma = 7, \Delta = 10$, on a $\mathcal{X} = S \cup P \cup A_S^\omega \cup A_S^\sigma \cup A_P^\omega$ et $|A_S^\omega| = 4$, $|A_P^\omega| = 4$, $|A_S^\sigma| = 44$, $|S| = 216$, $|P| = 216$ et enfin $|\mathcal{X}| = 484$.

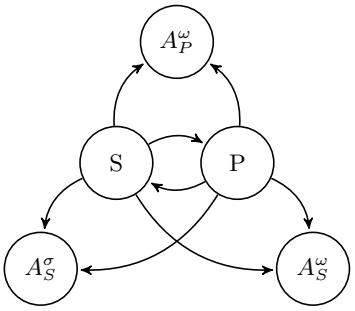


FIGURE 6.12 – Vue agrégée de la chaîne de Markov $X^{(\phi)}$. Le paramètre ϕ n'a aucune influence sur le nombre total d'états de $\mathcal{X}$. Dans le cas où $\gamma = 7$ et $\Delta = 7$, l'espace d'états contient 248 états.

$$M^{(\phi)} = \begin{pmatrix} M_S & M_{SP} & M_{SA_S^\omega} & M_{SA_S^\sigma} & M_{SA_P^\omega} \\ M_{PS} & M_P & M_{PA_S^\omega} & M_{PA_S^\sigma} & M_{PA_P^\omega} \\ 0 & 0 & M_{A_S^\omega} & 0 & 0 \\ 0 & 0 & 0 & M_{A_S^\sigma} & 0 \\ 0 & 0 & 0 & 0 & M_{A_P^\omega} \end{pmatrix}$$

avec M_{UV} la sous-matrice de dimension $|U| \times |V|$ contenant les probabilités de transition des états de U vers les états de V, avec $U, V \in \{S, P, A_S^\omega, A_S^\sigma, A_P^\omega\}$. On notera simplement M_U la sous-matrice M_{UU}.

De manière similaire, on note α, la distribution de probabilité initiale. Ce vecteur est partitionné de la manière suivante :

$$\alpha = (\alpha_S \;\; \alpha_P \;\; \alpha_{A_S^\omega} \;\; \alpha_{A_S^\sigma} \;\; \alpha_{A_P^\omega}),$$

avec α_U le sous-vecteur contenant les probabilités initiales des états de $U \in \{S, P, A_S^\omega, A_S^\sigma, A_P^\omega\}$.

Dans le cadre de la règle 1, l'adversaire provoque le départ volontaire d'une entité malveillante du core de $\mathcal{C}$ si ce départ lui permet d'augmenter sa représentation au sein du core après renouvellement. On modélise l'état $(s, x, y) \in \mathcal{X}$ d'un cluster $\mathcal{C}$ par deux urnes. La première contient γ boules, dont x sont rouges et $\gamma - x$ sont blanches. La seconde contient s boules, dont y sont rouges et $s - y$ sont blanches.

Le protocole de renouvellement P_ϕ consiste à sélectionner $\phi - 1$ entités du core qui sont placées dans le spare avant de sélectionner ϕ entités du spare élues pour intégrer le core. On modélise ce protocole P_ϕ par un tirage de $\phi - 1$ boules dans la première urne qui sont placées dans la seconde. Un deuxième tirage a lieu dans la seconde urne de ϕ boules qu'on replace ensuite dans la première.

Considérons une urne contenant ℓ boules dont v sont rouges et $\ell - v$ sont blanches. Pour tout $\ell \geq 0$, pour tout $0 \leq k \leq \ell$, pour tout $0 \leq v \leq \ell$ et tout $0 \leq u \leq v$, on note $q(k, \ell, u, v)$ la probabilité de tirer u boules rouges lors d'un tirage sans remise de

k boules dans cette urne. Cette probabilité suit une loi hypergéométrique, on a :

$$q(k,\ell,u,v) = \begin{cases} \frac{\binom{v}{u}\binom{\ell-v}{k-u}}{\binom{\ell}{k}} & \text{si } 0 \leq u \leq v, 0 \leq k \leq \ell, 0 \leq k-u \leq \ell - v \\ 0 & \text{sinon.} \end{cases}$$

Soit $\mathcal{C} = \{C, S\}$ un cluster dont le core C contient x entités malveillantes et le spare S contient y entités malveillantes, le calcul de la relation (6.1) s'effectue en deux temps. Dans le cas d'un départ volontaire d'une entité malveillante du core, on a $|C| = \gamma - 1$ et $x - 1$ entités malveillantes encore présentes dans C. On choisit dans un premier temps $\phi - 1$ entités du core à mettre dans le spare. On choisit ensuite ϕ entités dans le spare a remettre dans le core. On a donc renouvellé $\phi - 1$ entités du core, et celui-ci a retrouvé une taille γ. La relation (6.1) s'écrit alors

$$\begin{aligned} &\Pr\{|\{\text{entités malveillantes dans } C \text{ après renouvellement}\}| > x\} > 1 - \eta \\ &= \sum_{j=x+1}^{x-1+\min(\phi,y)} \Pr\left\{ \begin{array}{l} \text{exactement } j \text{ entités malveillantes} \in C \\ \text{après le renouvellement de } C \end{array} \right\} > 1 - \eta \\ &= \sum_{i=0}^{x-1} q(\phi-1, \gamma-1, i, x-1) \sum_{j=x+1}^{\min(\phi, y+i)} q(\phi, s+\phi-1, j, y+i) > 1 - \eta \end{aligned} \tag{6.2}$$

La règle 1 s'applique si on a :

— $x > 0$: au moins une entité malveillante doit être présente dans le core de $\mathcal{C}$ pour pouvoir déclencher un renouvellement de celui-ci,

— $y > 0$: au moins une entité malveillante doit être présente dans le spare de $\mathcal{C}$ pour pouvoir augmenter la représentation des entités malveillantes au sein du core,

— $\phi > 1$: il faut renouveler au minimum 2 entités du core de $\mathcal{C}$ pour pouvoir compenser le départ d'une entité malveillante et augmenter la représentation des entités malveillantes au sein du core,

— la somme des probabilités de transition vers les états $(s-1, x', y') \in \mathcal{X}$ avec $x' > x$ et y' tel que $x + y - 1 = x' + y'$ est supérieure à $1 - \eta$.

La relation (6.2) est utilisée pour le calcul des probabilités de transitions entre l'état (s, x, y) et les états $(s-1, x-a-b, y-a+b)$, $(s-1, x-1-a+b, y+a-b)$ et $(s-1, x-1-a+b, y+a-b)$, avec $0 \leq a \leq \min(\gamma, x+\phi), 0 \leq b \leq y+\phi$.

La matrice $M^{(\phi)}$ contient l'ensemble des probabilités de transitions. Les valeurs des coefficients de $M^{(\phi)}$ sont calculées au moyen de l'arbre décrit dans la figure 6.13 ainsi que dans le tableau 6.2. Dans cette figure, on représente les états atteignables depuis un état (s, x, y) ainsi que le calcul des probabilités de transitions pour chaque état atteignable. Ces probabilités de transitions dépendent :

i) du type d'événement impactant le cluster (`join` ou `leave`),

ii) du type d'entité impliqué dans cet événement (honnête ou malveillante),

iii) de la proportion d'entités malveillantes dans chacun des ensembles composant le cluster.

Ainsi, la probabilité de transition vers un état cible est obtenue en sommant le produit des transitions obtenues le long de chaque chemin allant de la racine de l'arbre à une feuille correspondant à l'état cible.

Dans ce diagramme on modélise la durée de validité des identifiants par la probabilité δ que cet identifiant soit encore valide au sens de la propriété 6. On suppose que les entités entrent dans le système de manière indépendante. Par conséquent, la probabilité que parmi un ensemble de z entités, toutes aient encore un identifiant valide au sens de la propriété 6 vaut δ^z. La probabilité que parmi un ensemble de z entités, au moins l'une d'elle ait un identifiant périmé vaut alors $1-\delta^z$. Cette valeur est utilisée pour traduire les départs imposés dus à la péremption de l'identifiant des entités malveillantes du cluster. Enfin, nous utilisons ici la notation "$1_{\{(*)\}}$" valant 1 lorsque la relation (*) est satisfaite et 0 sinon. Le tableau 6.2 fournit la signification des notations employées dans ce diagramme.

Par exemple, la branche bleue (en pointillés gras et en bas sur la figure 6.13) correspond au scénario dans lequel une entité malveillante souhaite intégrer le cluster $\mathcal{C}$. D'après la règle 2, cette entité est ajoutée au cluster dans le spare, amenant ainsi l'état du cluster à évoluer de (s, x, y) à $(s+1, x, y+1)$. Cette transition est effectuée avec la probabilité suivante :

$$p_j \times 1_{\{x>c\}} \times 1_{\{s<\Delta-1\}} \times 1_{\{s>1\}} \times p_m.$$

De manière similaire, la branche rouge (en gras et en haut sur la figure 6.13) représente la situation ou $\mathcal{C}$ est pollué et une des entités j malveillantes présente dans le core a un identifiant qui n'est plus valide. D'après la propriété 6, cette entité j doit quitter le cluster $\mathcal{C}$ pour se replacer dans un nouveau cluster $\mathcal{C}'$. Cependant, le core de $\mathcal{C}$ comporte encore suffisamment d'entités malveillantes ($x-1 > c$) pour que l'adversaire soit capable de biaiser le consensus de gestion du renouvellement du core. Ainsi, l'entité j est remplacée par une autre entité malveillante issue du spare. Dans ce cas, l'état du cluster $\mathcal{C}$ est alors $(s-1, x, y-1)$. Cette transition est effectuée avec la probabilité suivante :

$$p_l \times p_c \times p_{mc} \times (1-\delta^x) \times 1_{\{x-1>c\}} \times 1_{\{y>0\}}.$$

Les probabilités de transitions sont calculées à partir du diagramme 6.13. Ces probabilités constituent les coefficients de la matrice $M^{(\phi)}$. Pour tout $s \in \{1, \ldots, \Delta-1\}$, pour tout $x \in \{0, \ldots, \gamma\}$ et pour tout $y \in \{0, \ldots, s\}$, on a :

$$\begin{aligned}
p_{(s,x,y),(s,x,y)} &= p_j 1_{\{x>c\}} \left(1_{\{s=\Delta-1\}} + 1_{\{s<\Delta-1\}} 1_{\{s>1\}} (1-p_m)\right) \\
&\quad + \; p_\ell \left((1-p_c)\, p_{ms} \delta^y + p_c p_{mc} \delta^x \left(1_{\{x>c\}} + 1_{\{x\leq c\}} \left(1_{\{(\overline{6.2})\}} + 1_{\{s=1\}} 1_{\{(6.2)\}}\right)\right)\right) \\
p_{(s,x,y),(s+1,x,y)} &= p_j \left(1_{\{x>c\}} 1_{\{s<\Delta-1\}} 1_{\{s=1\}} + 1_{\{x\leq c\}}\right) (1-p_m)
\end{aligned}$$

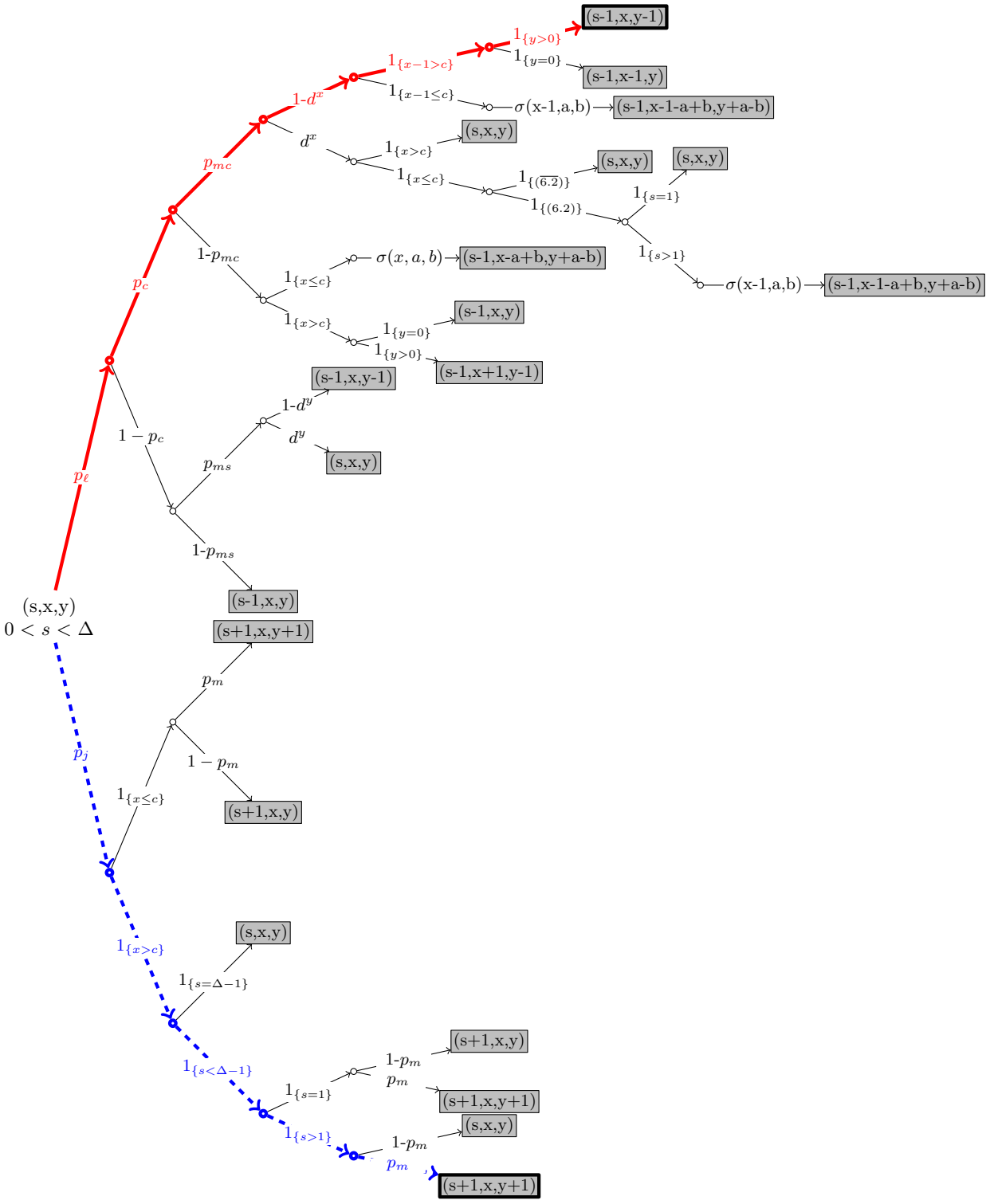


FIGURE 6.13 – Diagramme de transition pour le calcul des coefficients de la matrice de transition $M^{(r)}$.

TABLE 6.2 – Signification des notations employées dans la figure 6.13

Notation	Signification
μ , $0 \leq \mu < 1$	proportion d'entités malveillantes dans le système
γ	taille du core d'un cluster
Γ	taille maximale d'un cluster
$\Delta = \Gamma - \gamma$	taille maximale du spare d'un cluster
$c = \lfloor (\gamma - 1)/2 \rfloor$	nombre maximal d'entités malveillantes tolérées dans le core sans que le cluster soit pollué
s	taille du spare
x, $0 \leq x \leq \gamma$	nombre d'entités malveillantes dans le core
y, $0 \leq y \leq s$	nombre d'entités malveillantes dans le spare
ϕ, $1, \ldots, \gamma$	nombre d'entités du spare élues pour intégrer le core
δ, $0 \leq \delta < 1$	probabilité qu'un identifiant soit valide
$p_j = p_\ell = 1/2$	probabilité d'un événement `join` ou `leave`
$p_m = \mu$	probabilité que l'entité entrante soit malveillante
$p_c = \gamma/(\gamma + s)$	probabilité qu'une entité appartienne au core du cluster
$p_{mc} = x/\gamma$	probabilité qu'une entité du core soit malveillante
$p_{ms} = y/s$	probabilité qu'une entité du spare soit malveillante
δ^x	probabilité que la propriété 6 soit satisfaite dans le core
δ^y	probabilité que la propriété 6 soit satisfaite dans le spare
$1_{\{A\}}$	fonction indicatrice, retourne 1 si A est vraie, 0 sinon
$\sigma(x, a, b)$	probabilité de contruire un core contenant $x - a + b$ entités malveillantes et un spare en contenant $y - b + a$. $\sigma(x, a, b) = q(\phi - 1, \gamma - 1, a, x)q(\phi, s + \phi - 1, b, y + a)$ $0 \leq a \leq \min(\gamma, x + \phi), 0 \leq b \leq y + \phi$

$$p_{(s,x,y),(s+1,x,y+1)} = p_j \left(1_{\{x>c\}} 1_{\{s<\Delta-1\}} 1_{\{s\geq 1\}} + 1_{\{x\leq c\}}\right) p_m$$

$$p_{(s,x,y),(s-1,x,y-1)} = p_\ell \left((1-p_c)p_{ms}(1-\delta^y) + p_c p_{mc} \left(\delta^x 1_{\{x\leq c\}} 1_{\{(6.2)\}} 1_{\{s>1\}} \sigma(x-1,a,b)\right.\right.$$
$$\left.\left. + \ (1-\delta^x)\left(1_{\{x-1\leq c\}}\sigma(x-1,a,b) + 1_{\{x-1>c\}}1_{\{y>0\}}\right)\right)\right) \text{ pour } b-a=1$$

$$p_{(s,x,y),(s-1,x-1,y)} = p_\ell p_c p_{mc}(\delta^x 1_{\{x\leq c\}} 1_{\{(6.2)\}} 1_{\{s>1\}} \sigma(x-1,a,b)$$
$$+ (1-\delta^x) 1_{\{x-1>c\}} 1_{\{y=0\}}) \text{ pour } b-a=0$$

$$p_{(s,x,y),(s-1,x,y)} = p_\ell \left((1-p_c)(1-p_{ms}) + p_c(1-p_{mc})\left(1_{\{x>c\}}1_{\{y=0\}} + 1_{\{x\leq c\}}\sigma(x,a,b)\right)\right)$$
$$\text{pour } b-a=0$$

$$p_{(s,x,y),(s-1,x+1,y-1)} = p_\ell p_c (1-p_{mc})\left(1_{\{x>c\}}1_{\{y>0\}} + 1_{\{x\leq c\}}\sigma(x,a,b)\right) \text{ pour } b-a=1$$

$$p_{(s,x,y),(s-1,x-a+b,y+a-b)} = p_\ell p_c (1-p_{mc}) 1_{\{x\leq c\}}\sigma(x,a,b)$$
$$\text{et } \max(0,\phi-\gamma+x) \leq a \leq \min(x,\phi-1)$$
$$\text{et } \max(0,y+a-s-1) \leq b \leq \min(y+a,\phi) \text{ pour } b-a \notin \{0,1\}$$

$$p_{(s,x,y),(s-1,x,y-1)} = p_\ell p_c p_{mc}\left(\delta^x 1_{\{x\leq c\}} 1_{\{(6.2)\}} 1_{\{s>1\}} + (1-\delta^x) 1_{\{x-1\leq c\}}\right)\sigma(x-1,a,b)$$
$$\text{et } \max(0,\phi-1-\gamma+x) \leq a \leq \min(x-1,\phi-1)$$
$$\text{et } \max(0,y+a-s-1) \leq b \leq \min(y+a,\phi) \text{ pour } b-a \notin \{0,1\}$$

Dans les autres cas, les probabilités de transitions sont nulles. Nous allons utiliser cette matrice dans la section suivante afin d'étudier l'évolution de la composition d'un cluster.

6.4 Étude de la composition d'un cluster

Nous avons modélisé dans la section précédente la composition d'un cluster au moyen d'une chaîne de Markov à temps discret. Dans cette section, nous utilisons cette modélisation afin d'étudier le comportement du cluster vis à vis de

- la puissance μ de l'adversaire,
- la durée de validité des identifiants, représentée par δ,
- et la quantité ϕ d'entités renouvelées lors d'une mise à jour du core du cluster suite à un départ d'une entité de celui-ci.

Ainsi, dans cette section nous étudions le temps passé dans les états sains et les états pollués pour chaque protocole P_ϕ, les fréquences d'alternances entre ces états et enfin l'éventuelle propagation de la pollution d'un cluster à d'autres parties du système.

Nous considérons deux distributions initiales $\alpha^{(1)}$ et $\alpha^{(2)}$. La première correspond à une situation où le cluster ne contient aucune entité malveillante et le spare a pour une taille $\Delta/2$. On a :

$$\alpha^{(1)}_{(s,x,y)} = \begin{cases} 1 & \text{si } (s,x,y) = (\Delta/2,0,0) \\ 0 & \text{sinon.} \end{cases}$$

La seconde distribution initiale que nous considérons correspond à un cluster dont le spare a pour une taille $\Delta/2$ et la répartition des entités malveillantes dans le spare et

le core suit une distribution binomiale. On a alors :

$$\alpha^{(2)}_{(s,x,y)} = \begin{cases} \binom{\gamma}{x}\mu^x(1-\mu)^{\gamma-x}\binom{\Delta/2}{y}\mu^y(1-\mu)^{\Delta/2-y} & \text{pour } (\Delta/2,x,y) \in S \cup P \\ 0 & \text{sinon.} \end{cases}$$

6.4.1 Espérance du temps total passé dans les états sains et pollués

On s'intéresse tout d'abord au temps passé par le cluster dans les états sains avant que celui-ci ne soit amené à se scinder en deux nouveaux clusters ou à devoir fusionner avec un autre. Comme nous l'avons vu dans la section précédente, la chaîne de Markov $X^{(r)}$ est absorbante : les états de S et P sont transitoires et les états de A^{ω}_S, A^{σ}_S et A^{ω}_P sont absorbants. On définit alors la variable aléatoire $T^{(\phi)}_S$ qui compte le temps total passé dans les états sains avant absorption par un cluster appliquant le protocole P_ϕ. On a alors

$$T^{(\phi)}_S = \sum_{m=0}^{\infty} 1_{\{X^{(\phi)}_m \in S\}}.$$

D'après [Ser90], la loi de $T^{(\phi)}_S$ est donnée par :

$$\Pr\{T^{(\phi)}_S = \ell\} = \begin{cases} 1 - v\mathbb{1} & \text{si } \ell = 0 \\ vR^{\ell-1}(I-R)\mathbb{1} & \text{si } \ell \geq 1 \end{cases} \tag{6.3}$$

où I désigne la matrice identité, $v = \alpha_S + \alpha_P(I - M_P)^{-1}M_{PS}$, $R = M_S + M_{SP}(I - M_P)^{-1}M_{PS}$ et où $\mathbb{1}$ désigne le vecteur colonne de dimension ad-hoc ne contenant que des 1. La fonction de répartition et l'espérance de $T^{(\phi)}_S$ sont données par les relations suivantes :

$$\Pr\{T^{(\phi)}_S \leq \ell\} = 1 - vR^{\ell}\mathbb{1} \quad \text{et} \quad E(T^{(\phi)}_S) = v(I-R)^{-1}\mathbb{1}. \tag{6.4}$$

De manière similaire, on définit la variable aléatoire suivante :

$$T^{(\phi)}_P = \sum_{m=0}^{\infty} 1_{\{X^{(\phi)}_m \in P\}}$$

Cette variable compte le temps total passé dans les états pollués avant absorption par un cluster appliquant le protocole P_ϕ. La loi de $T^{(\phi)}_P$ est donnée par la relation (6.5), sa fonction de répartition et son espérance par la relation (6.6).

$$\Pr\{T^{(\phi)}_P = \ell\} = \begin{cases} 1 - w\mathbb{1} & \text{si } \ell = 0 \\ wQ^{\ell-1}(I-Q)\mathbb{1} & \text{si } \ell \geq 1 \end{cases} \tag{6.5}$$

$$\Pr\{T^{(\phi)}_P \leq \ell\} = 1 - wQ^{\ell}\mathbb{1} \text{ et } E(T^{(\phi)}_P) = w(I-Q)^{-1}\mathbb{1}, \tag{6.6}$$

où I désigne la matrice identité, $w = \alpha_P + \alpha_S(I - M_S)^{-1}M_{SP}$ et $Q = M_P + M_{PS}(I - M_S)^{-1}M_{SP}$.

Les figures 6.14 et 6.15 illustrent l'espérance du temps passé dans les états sains $E(T_S^{(\phi)})$ et les états pollués $E(T_P^{(\phi)})$ avant absorption en nombre d'événements `join` et `leave` pour différentes proportions μ d'entités malveillantes, et différentes probabilités δ qu'un identifiant soit valide. Les figures 6.14(a) et 6.15(a) illustrent ces quantités pour $\phi = 1$ pour les distributions initiales $\alpha^{(1)}$ et $\alpha^{(2)}$ considérées. Les figures 6.14(b) et 6.15(b) illustrent ces quantités pour $\phi = \gamma$ pour les distributions initiales $\alpha^{(1)}$ et $\alpha^{(2)}$ considérées.

Ces deux protocoles représentent les deux situations extrêmes concernant ϕ. Dans le premier cas, lors du départ d'une entité du core, une seule entité est choisie parmi le spare pour remplacer l'entité partie, tandis que dans le second cas, l'ensemble du core est renouvelé en choisissant aléatoirement un sous-ensemble de taille γ parmi les entités du cluster.

Tout d'abord, quelle que soit la distribution initiale ou la proportion d'entités renouvelées dans la mise à jour du core, on constate qu'à durée de vie fixée, la proportion de temps passé dans les états pollués augmente avec la proportion d'entités malveillantes dans le système. En effet, plus cette proportion augmente, plus la probabilité d'ajouter une entité malveillante au cluster augmente, et donc la probabilité que celle-ci soit choisie pour faire partie du core.

D'autre part, on constate que le temps passé dans les états pollués croît avec δ. En effet, l'adversaire laissant ses entités malveillantes le plus longtemps possible au sein du cluster, plus leur durée de validité augmente, plus la probabilité d'être choisie pour faire partie du core augmente. D'autre part, l'adversaire bloque les arrivées d'entités honnêtes (règle 2), limitant ainsi la possibilité de revenir à un état sain.

Le tableau 6.3 illustre cette tendance pour valeurs de δ proches de 1. La durée de validité des identifiants tend vers l'infini lorsque δ s'approche de 1. On constate alors que lorsque δ tend vers 1, la classe d'états P tend à devenir absorbante. Ainsi pour $\delta = 0.999$, dès $\mu = 10\%$ le temps moyen passé dans les états sains avoisine les 700000 événements de `join` et `leave` alors qu'en l'absence d'entités malveillantes, la durée de vie moyenne du cluster n'est que de 12 événements.

D'autre part, on constate que la distribution initiale d'états du cluster a un impact sur l'espérance du temps passé dans les états sains et celle du temps passé dans les états pollués. En effet, dans le cas de la distribution initiale $\alpha^{(1)}$, le cluster est initialement exempt d'entités malveillantes. Par conséquent, les états contenant des entités malveillantes ne sont pas immédiatement accessibles. Le cluster vit alors en état sain et fini par atteindre un état où il doit se scinder en deux ou fusionner avec un autre cluster avant qu'il n'y ait suffisamment d'entités malveillantes au sein du cluster permettant de prendre le contrôle de celui-ci et ainsi de bloquer l'arrivée d'autres entités honnêtes.

À l'inverse, dans le cas de la distribution initiale $\alpha^{(2)}$, des entités malveillantes sont déjà potentiellement présentes dans le cluster. Lorsque l'adversaire prend le contrôle du cluster, celui-ci bloque les arrivées d'entités honnêtes augmentant ainsi significativement l'espérance du temps passé dans les états pollués.

Enfin, on constate sur cette figure que le choix du protocole a un impact sur l'espérance des temps totaux passés dans les états sains et pollués. Étonnamment, le protocole P_1 offre une espérance du temps total passé dans les états sains nettement

TABLE 6.3 – $E(T_S^{(\phi)})$ et $E(T_P^{(\phi)})$ en fonction de μ et δ dans le cas où $\phi = 1$, $\gamma = 7$, $\Delta = 7$ et $\alpha = \alpha^{(1)}$

	$\mu = 0\%$		$\mu = 10\%$		$\mu = 20\%$		$\mu = 30\%$	
δ	0.9	0.999	0.9	0.999	0.9	0.999	0.9	0.999
$E(T_S^{(1)})$	12.0	12.0	12.18	12.19	12.21	12.16	12.12	11.99
$E(T_P^{(1)})$	0.0	0.0	0.02	698591	0.17	2.57×10^8	0.64	4.92×10^9

plus faible que dans le cas du protocole P_γ. Ceci est dû à deux facteurs combinés : la quantité ϕ d'entités du core renouvelées lors du départ d'une entité du core et l'application par l'adversaire de la règle 2.

P_1 : Dans ce cas, les entités malveillantes intègrent le core une par une. Il faut donc au minimum $c+1$ départ d'entités du core pour y insérer les entités malveillantes nécessaires à la prise de contrôle du cluster par l'adversaire. Dans le cas de la distribution initiale $\alpha^{(1)}$, il faut de plus au minimum $c + 1$ ajouts d'entités malveillantes dans le cluster pour permettre la pollution.

P_γ : Si le cluster comporte $c + 1$ entités, la pollution est possible dès le premier départ d'une entité du core. De plus, d'après la règle 2, l'adversaire ignore tout événement `join` d'une entité honnête. Il augmente ainsi la représentation des entités malveillantes au sein du cluster de manière significative et donc l'espérance du temps passé dans les états pollués.

6.4.2 Temps successifs passés dans les états sains et pollués

Les variables $T_S^{(\phi)}$ et $T_P^{(\phi)}$ représentent le temps total passé dans les états sains ou pollués avant l'absorption. Cependant, ces variables ne caractérisent pas l'alternance entre ces états. On définit alors les variables aléatoires suivantes $T_{S,m}^{(\phi)}$ et $T_{P,m}^{(\phi)}$ représentant la durée du m-ème séjour de $X^{(\phi)}$ dans S (respectivement dans P) pour $m \geq 1$. Le temps total passé dans les états sains (respectivement pollué) est donné par :

$$T_S^{(\phi)} = \sum_{m=1}^{\infty} T_{S,m}^{(\phi)} \quad \text{et} \quad T_P^{(\phi)} = \sum_{m=1}^{\infty} T_{P,m}^{(\phi)}.$$

D'après [RS89], on a pour tout $m \geq 1$ et tout $\ell \geq 0$

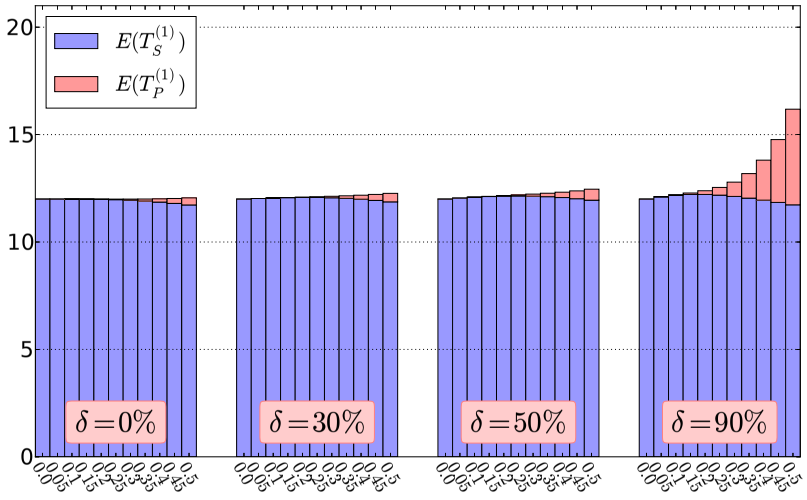
$$\Pr\{T_{S,m}^{(\phi)} \leq \ell\} = 1 - vG^{m-1}(M_S)^\ell \mathbb{1}, \tag{6.7}$$

où v est défini par la relation (6.4) et $G = (I - M_S)^{-1}M_{SP}(I - M_P)^{-1}M_{PS}$.

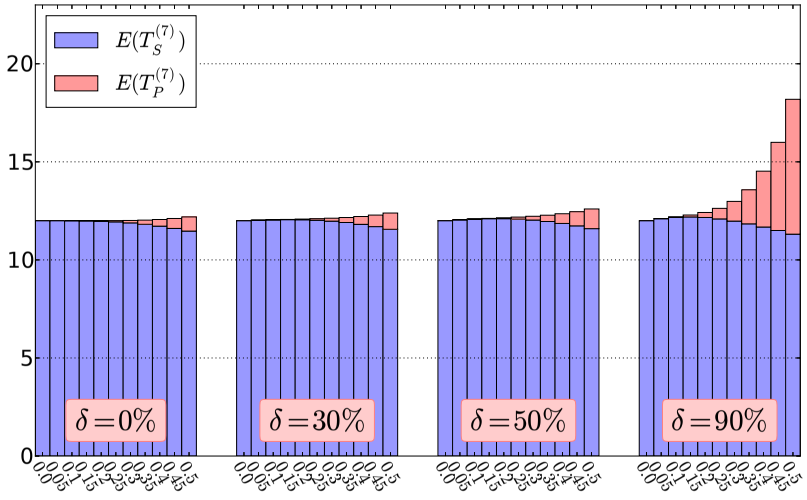
De manière symétrique, on a

$$\Pr\{T_{P,m}^{(\phi)} \leq \ell\} = 1 - wH^{m-1}(M_P)^\ell \mathbb{1}, \tag{6.8}$$

où w défini par la relation (6.6) et $H = (I - M_P)^{-1}M_{PS}(I - M_S)^{-1}M_{SP}$.

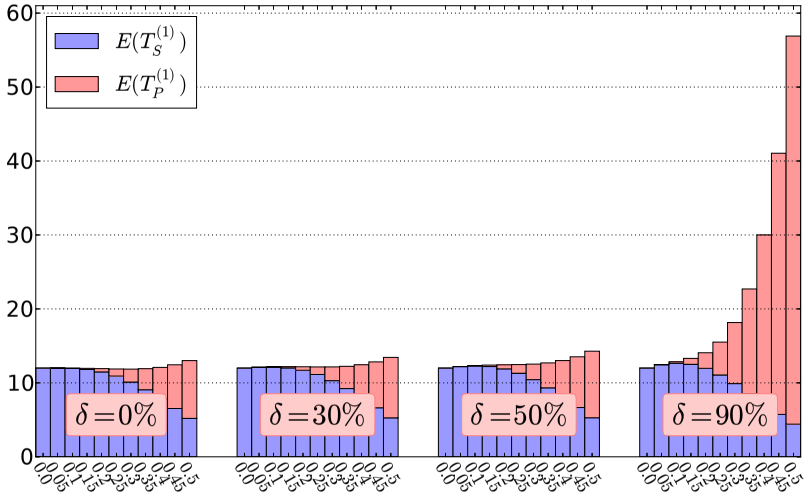


(a) $\phi = 1, \alpha = \alpha^{(1)}$

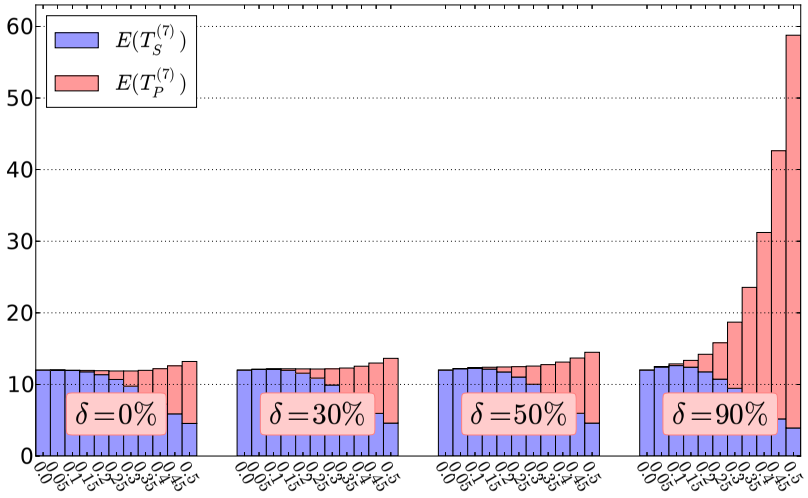


(b) $\phi = 7, \alpha = \alpha^{(1)}$

FIGURE 6.14 – $E(T_S^{(\phi)})$ (Relation (6.4)) et $E(T_P^{(\phi)})$ (Relation (6.6)) en fonction de ϕ, δ et μ dans le cas où $\alpha = \alpha^{(1)}$, $\gamma = 7, \Delta = 7$ et $\eta = 0.2$.



(a) $\phi = 1, \alpha = \alpha^{(2)}$



(b) $\phi = 7, \alpha = \alpha^{(2)}$

FIGURE 6.15 – $E(T_S^{(\phi)})$ (Relation (6.4)) et $E(T_P^{(\phi)})$ (Relation (6.6)) en fonction de ϕ, δ et μ dans le cas où $\alpha = \alpha^{(2)}$, $\gamma = 7, \Delta = 7$ et $\eta = 0.2$.

TABLE 6.4 – Durées successives dans les états de S et P en fonction μ pour $\phi = 1$, $\gamma = 7$, $\Delta = 7$, $\delta = 90\%$, et $\alpha = \alpha^{(1)}$.

	$\mu = 0\%$	$\mu = 10\%$	$\mu = 20\%$	$\mu = 30\%$	$\mu = 40\%$	$\mu = 50\%$
$E(T_{S,1}^{(1)})$	12.0	12.17	12.19	12.08	11.90	11.68
$E(T_{S,2}^{(1)})$	0.0	0.00	0.02	0.03	0.03	0.044
$E(T_{P,1}^{(1)})$	0.0	0.01	0.15	0.61	1.72	4.16
$E(T_{P,2}^{(1)})$	0.0	0.0	0.01	0.04	0.12	0.26

L'espérance de la durée du m-ème séjour passé dans S et dans P est donnée par les relations suivantes :

$$E(T_{S,m}^{(\phi)}) = vG^{m-1}(I - M_S)^{-1}\mathbb{1} \tag{6.9}$$

$$E(T_{P,m}^{(\phi)}) = wH^{m-1}(I - M_P)^{-1}\mathbb{1}. \tag{6.10}$$

Le tableau 6.4 illustre ces temps moyens dans le cas du protocole P_1. On constate alors qu'on a $E(T_S^{(r)}) \simeq E(T_{S,1}^{(r)})$ et $E(T_P^{(r)}) \simeq E(T_{P,1}^{(r)})$. Cela signifie que dans le cas du protocole P_1, il n'y a quasiment pas d'alternance entre les états de S et de P : partant d'un état sain, le cluster fini par se scinder ou fusionner avec un autre avec très peu de passage par des états pollués. Même dans le cas de durées de validité assez longues ($\delta = 90\%$), la dynamique du système prime sur la capacité de l'adversaire à polluer le cluster.

6.4.3 Probabilités d'absorption

Nous venons d'étudier le comportement d'un cluster vis-à-vis des entrées et des sorties d'entités de celui-ci en présence d'un adversaire visant à manipuler le système. On s'intéresse à présent à l'impact de cet adversaire sur les opérations `split` et `merge` du cluster. En effet, ces opérations influent sur le nombre de clusters présents dans le système, il est donc intéressant de caractériser l'influence de l'adversaire sur le système afin d'évaluer l'éventuelle propagation de la pollution au sein du système.

On a la matrice $M^{(\phi)}$ suivante :

$$\begin{pmatrix} M_S & M_{SP} & M_{SA_S^\omega} & M_{SA_S^\sigma} & M_{SA_P^\omega} \\ M_{PS} & M_P & M_{PA_S^\omega} & M_{PA_S^\sigma} & M_{PA_P^\omega} \\ 0 & 0 & M_{A_S^\omega} & 0 & 0 \\ 0 & 0 & 0 & M_{A_S^\sigma} & 0 \\ 0 & 0 & 0 & 0 & M_{A_P^\omega} \end{pmatrix}$$

On redéfinit le découpage en sous-matrices de la matrices $M^{(\phi)}$ comme suit :

$$\begin{pmatrix} & T^{(\phi)} & R_S^\omega & R_S^\sigma & R_P^\omega \\ \boxed{\begin{matrix} M_S & M_{SP} \\ M_{PS} & M_P \end{matrix}} & & \boxed{\begin{matrix} M_{SA_S^\omega} \\ M_{PA_S^\omega} \end{matrix}} & \boxed{\begin{matrix} M_{SA_S^\sigma} \\ M_{PA_S^\sigma} \end{matrix}} & \boxed{\begin{matrix} M_{SA_P^\omega} \\ M_{PA_P^\omega} \end{matrix}} \\ 0 & 0 & M_{A_S^\omega} & 0 & 0 \\ 0 & 0 & 0 & M_{A_S^\sigma} & 0 \\ 0 & 0 & 0 & 0 & M_{A_P^\omega} \end{pmatrix}$$

$$M^{(\phi)} = \begin{pmatrix} T^{(\phi)} & R_S^\omega & R_S^\sigma & R_P^\omega \\ 0 & * & * & * \end{pmatrix} \text{ avec } T^{(r)} = \begin{pmatrix} M_S & M_{SP} \\ M_{PS} & M_P \end{pmatrix},$$
$$R_S^\omega = \begin{pmatrix} M_{SA_S^\omega} \\ M_{PA_S^\omega} \end{pmatrix}, R_S^\sigma = \begin{pmatrix} M_{SA_S^\sigma} \\ M_{PA_S^\sigma} \end{pmatrix} \text{ et } R_P^\omega = \begin{pmatrix} M_{SA_P^\omega} \\ M_{PA_P^\omega} \end{pmatrix}.$$

Ainsi, la probabilité $p(A_S^\sigma)$ que la chaîne de Markov $X^{(\phi)}$ soit absorbée dans la classe A_S^σ est donnée par

$$p(A_S^\sigma) = \alpha_{T^{(\phi)}}(I - T^{(\phi)})^{-1} R_S^\sigma \mathbb{1} \text{ où } \alpha_{T^{(\phi)}} = (\alpha_S \ \alpha_P). \tag{6.11}$$

avec α_S le sous-vecteur de α correspondant aux états de S et α_P le sous-vecteur de α correspondant aux états de P.

De manière similaire, on calcule les probabilités $p(A_S^\omega)$ et $p(A_P^\omega)$ d'absorption de $X^{(\phi)}$ dans les classes A_S^ω et A_P^ω :

$$p(A_S^\omega) = \alpha_{T^{(\phi)}}(I - T^{(\phi)})^{-1} R_S^\omega \mathbb{1} \text{ où } \alpha_{T^{(\phi)}} = (\alpha_S \ \alpha_P). \tag{6.12}$$

$$p(A_P^\omega) = \alpha_{T^{(\phi)}}(I - T^{(\phi)})^{-1} R_P^\omega \mathbb{1} \text{ où } \alpha_{T^{(\phi)}} = (\alpha_S \ \alpha_P). \tag{6.13}$$

Les figure 6.16 et 6.17 illustrent les probabilités d'absorption $p(A_S^\sigma)$, $p(A_S^\omega)$ et $p(A_P^\omega)$ en fonction de la durée de validité des identifiants représentée par la probabilité δ et de la proportion d'entités malveillantes μ. Les figures 6.16(a) et 6.17(a) correspondent au protocole P_1 et les figures 6.16(b) et 6.17(b) au protocole P_γ.

On s'aperçoit tout d'abord en comparant les figures 6.16(a) et 6.16(b) que le protocole P_ϕ employé n'a pas d'incidence sur les probabilités d'absorption. En effet, l'adversaire agit sur la durée de vie du cluster lorsque celui-ci est pollué. Or l'absorption de la chaîne $X^{(\phi)}$ n'intervient que quand le cluster est sain dans le cas de la classe A_S^σ du fait de la règle 2 ou quand le cluster devient sous-peuplé dans les deux autres cas.

Les entités honnêtes peuvent quitter le cluster avant l'expiration de leur identifiant tandis que les entités malveillantes ne le quittent qu'à l'expiration de celui-ci. Par conséquent, l'adversaire n'a aucun contrôle sur ces départs et le taux de renouvellement du core n'a donc pas d'influence sur l'entrée dans une des classes absorbantes A_S^ω ouA_P^ω.

D'autre part, on constate qu'à proportion μ d'entité malveillantes fixée, le cluster a plus tendance à être absorbé dans la classe A_S^σ que dans les deux autres classes

absorbantes. En effet, les entités malveillantes effectuent moins d'opérations `leave` que d'opérations `join`, brisant ainsi l'équilibre entre ces opérations et induisant une dérive de la taille du cluster.

Enfin, on constate que dans tous les cas, la probabilité $p(A_P^\omega)$ est très faible, même pour de grandes proportions d'entités malveillantes. Ce dernier point est très important car il illustre le fait que la pollution d'un cluster ne s'étend pas à d'autres clusters du système et ainsi la probabilité pour un cluster d'être initialisé dans un état pollué est très faible.

6.5 Conclusion

Nous avons décrit dans ce chapitre l'architecture de PeerCube ainsi que les différentes opérations nécessaires à son fonctionnement. Nous avons modélisé la composition d'un cluster de PeerCube en présence d'un adversaire cherchant à manipuler le système. Nous avons modélisé l'évolution de la composition d'un cluster au moyen d'une chaîne de Markov a temps discret.

Cette modélisation nous a permis de déterminer en fonction du protocole employé P_ϕ, de la proportion μ d'entités malveillantes contrôlées par l'adversaire et de la durée de vie des identifiants les durées pendant lesquelles un cluster est sain ou pollué. De plus, nous avons pu analyser l'influence de l'adversaire sur le cluster et ainsi montrer que la pollution reste très majoritairement cloisonnée au cluster.

Nous étudions dans le chapitre suivant l'impact de la présence de l'adversaire sur la dynamique du système, évaluer sa capacité à polluer différents clusters du système et enfin à mettre en péril les requètes `lookup` effectuées dans PeerCube ainsi que des contres mesures pour limiter l'action de l'adversaire.

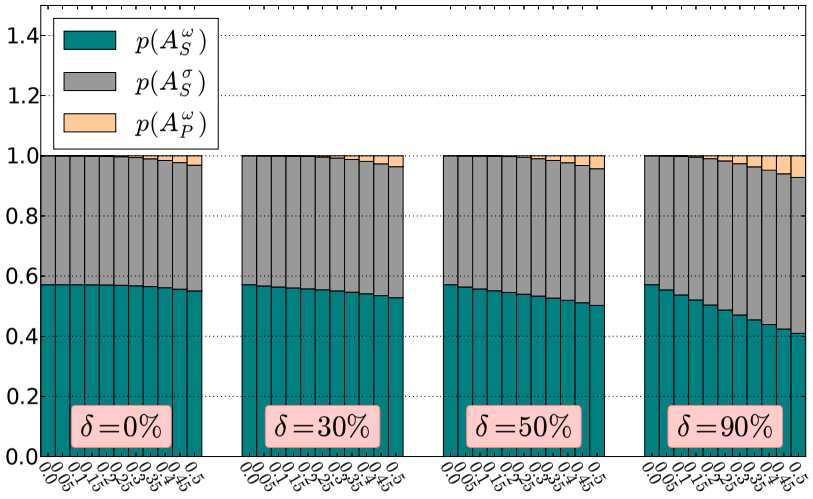
Bibliographie

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

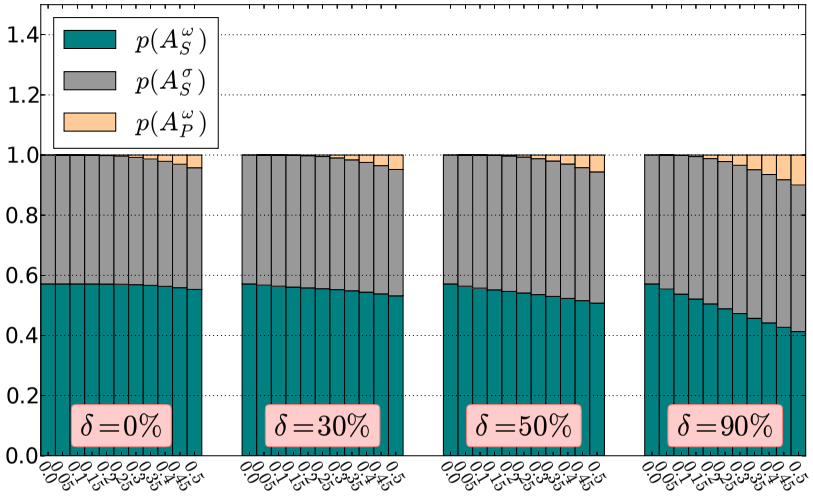
[AS07] B. Awerbuch et C. Scheideler : Towards Scalable and Robust Overay Networks. Dans *Proceedings of the International Workshop on Peer-to-Peer Systems*, IPTPS, 2007.

[ASLT11] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel : Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, juin 2011.

[EJ01] D. Eastlake et P. Jones : US Secure Hash Algorithm 1 (SHA1). Rapport technique, IETF, 2001.

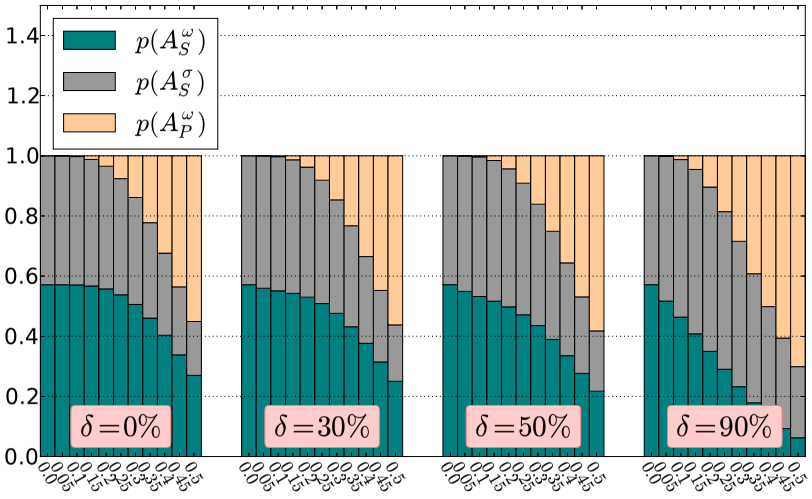


(a) $\phi = 1, \alpha = \alpha^{(1)}$

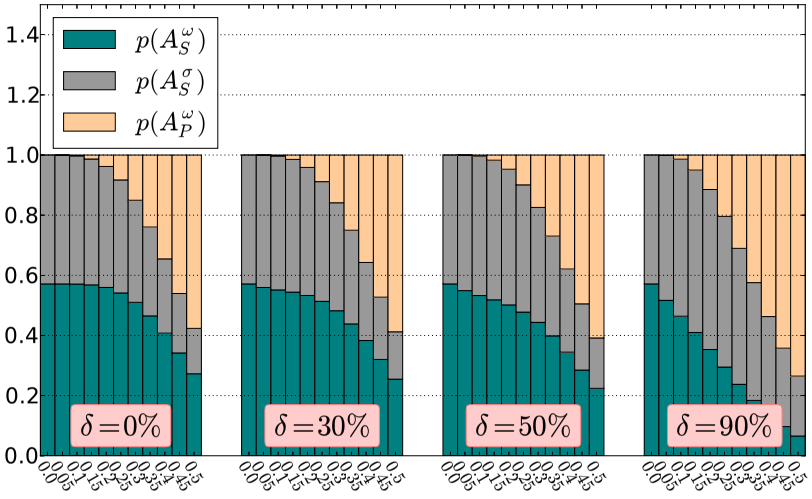


(b) $\phi = 7, \alpha = \alpha^{(1)}$

FIGURE 6.16 – Probabilités d'absorption $p(A_S^\omega)$, $p(A_S^\sigma)$ et $p(A_P^\omega)$ en fonction de ϕ, δ et μ dans le cas où $\alpha = \alpha^{(1)}$, $\gamma = 7$ et $\Delta = 7$.



(a) $\phi = 1, \alpha = \alpha^{(2)}$



(b) $\phi = 7, \alpha = \alpha^{(2)}$

FIGURE 6.17 – Probabilités d'absorption $p(A_S^\omega)$, $p(A_S^\sigma)$ and $p(A_P^\omega)$ en fonction de ϕ, δ et μ dans le cas où $\alpha = \alpha^{(2)}$, $\gamma = 7$ et $\Delta = 7$.

[HFPS99] R. Housley, W. Ford, W. Polk et D. Solo : Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile. Rapport technique, IETF, 1999.

[MMBK10] D. Mills, J. Martin, J. Burbank et W. Kasch : Network Time Protocol Version 4 : Protocol and Algorithms Specification. Rapport technique, IETF, 2010.

[RS89] G. Rubino et B. Sericola : Sojourn times in Markov processes. *Journal of Applied Probability*, 26(4):744–756, 1989.

[Ser90] B. Sericola : Closed form solution for the distribution of the total time spent in a subset of states of a Markov process during a finite observation period. *Journal of Applied Probability*, 27(3):713–719, 1990.

[SL04] M. Srivatsa et L. Liu : Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems : A quantitiative Analysis. Dans *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC, 2004.

[SS88] Y. Saad et M. Schultz : Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7), 1988.

Chapitre 7

Évaluation de PeerCube

Nous avons étudié au chapitre précédent l'évolution de la composition d'un cluster en présence d'un adversaire en fonction des entrées et des sorties des entités de celui-ci. Nous étudions à présent la dynamique du système dans son ensemble, et nous nous focalisons plus particulièrement sur deux aspects : le nombre de clusters sains et pollués dans le système et les changements topologiques au sein de celui-ci.

7.1 Étude de l'overlay

On considère un système composé de υ clusters $\mathcal{C}_1, \ldots, \mathcal{C}_\upsilon$ dans lequel chaque cluster $\mathcal{C}_j$ implémente le même protocole P_ϕ. De plus, on considère que les événements `join` et `leave` sont uniformément distribués sur l'overlay. Cela signifie que ces événements impactent uniformément les clusters et donc que pour chaque événement `join` ou `leave`, celui-ci impacte le cluster $\mathcal{C}_j, 1 \leq j \leq \upsilon$ avec probabilité $1/\upsilon$.

On associe à chaque cluster $\mathcal{C}_j, 1 \leq j \leq \upsilon$, la chaîne de Markov $X^{(\phi,j)}$ identique à $X^{(\phi)}$. On considère alors à présent un ensemble de υ chaînes de Markov notées $X^{(\phi,1)}, \ldots, X^{(\phi,\upsilon)}$ identiques à $X^{(\phi)}$. Ces chaînes ont le même espace d'état $\mathcal{X}$, la même matrice de transition $M^{(\phi)}$ et la même distribution initiale α.

Nous étudions à présent l'ensemble des chaînes $X^{(\phi,1)}, \ldots, X^{(\phi,\upsilon)}$. Dans ce cas, celles-ci ne sont cependant pas indépendantes. En effet, à chaque événement du système, une unique chaîne $X^{(\phi,j)}, 1 \leq j \leq \upsilon$ est choisie pour effectuer une transition. Cette chaîne est choisie avec la probabilité $1/\upsilon$.

On définit les variables aléatoires $N_S^{(\phi,\upsilon)}(m)$ et $N_P^{(\phi,\upsilon)}(m)$ comptant le nombre de clusters sains (respectivement pollués) de $\{\mathcal{C}_1, \ldots, \mathcal{C}_\upsilon\}$ juste après le m-ème événement `join` ou `leave`. La variable aléatoire $N_S^{(\phi,\upsilon)}(m)$ compte alors le nombre de chaînes $X^{(\phi,j)}, 1 \leq j \leq \upsilon$ dans un état sain à l'instant m, et $N_P^{(\phi,\upsilon)}(m)$ compte le nombre de chaînes $X^{(\phi,j)}, 1 \leq j \leq \upsilon$ dans un état pollué à l'instant m. Ces variables sont définies par :

$$N_S^{(\phi,\upsilon)}(m) = \sum_{j=1}^{\upsilon} 1_{\{X_m^{(\phi,j)} \in S\}} \text{ et } N_P^{(\phi,\upsilon)}(m) = \sum_{j=1}^{\upsilon} 1_{\{X_m^{(\phi,j)} \in P\}}.$$

Enfin, on note $\mathbb{1}_S$ le vecteur colonne de dimension $|S \cup P|$ dont la composante i vaut 1 si l'état i appartient à S et 0 s'il appartient à P. On définit de la même manière $\mathbb{1}_P$ le vecteur colonne de dimension $|S \cup P|$ dont la composante i vaut 1 si l'état i appartient à P et 0 s'il appartient à S. Nous étudions ces variables aléatoires dans un système de υ clusters exécutant le protocole P_ϕ.

Le système est composé de υ clusters $\mathcal{C}_1, \ldots, \mathcal{C}_\upsilon$ implémentant le même protocole P_ϕ. Dans un soucis de lisibilité, nous simplifions donc les notations ainsi : $X^{(\phi)}$, $X_m^{(\phi)}$, $X^{(\phi,j)}$, $X_m^{(\phi,j)}$, $M^{(\phi)}$ et $T^{(\phi)}$ sont ainsi notées X, X_m, $X^{(j)}$, $X_m^{(j)}$, M et T pour tout $j \in [\![1, \upsilon]\!]$. De plus, les variables $N_S^{(\phi,\upsilon)}(m)$ et $N_P^{(\phi,\upsilon)}(m)$ sont notées $N_S(m)$ et $N_P(m)$.

7.1.1 Espérance de la proportion de clusters sains et pollués

Nous calculons à présent l'espérance de $N_S(m)$ et $N_P(m)$. Nous avons montré dans [ACLS13] que la loi de chacune des variables aléatoires $X^{(j)}$ pour $j = 1, \ldots, \upsilon$ est donnée par le théorème suivant.

Théorème 9 ([ACLS13]) *Pour tout $j = 1, \ldots, \upsilon$, $m \geq 0$ et $(s, x, y) \in \mathcal{X}$, on a*

$$\Pr\{X_m^{(j)} = (s,x,y)\} = \sum_{\ell=0}^{m} \binom{m}{\ell} \left(\frac{1}{\upsilon}\right)^{\ell} \left(1 - \frac{1}{\upsilon}\right)^{m-\ell} \Pr\{X_\ell = (s,x,y)\} \tag{7.1}$$

Cette loi est indépendante de j. En effet, bien que ces chaînes de Markov aient une dépendance entre elles, chacune d'elle se comporte de manière identique et est choisie pour effectuer une transition avec une probabilité $1/\upsilon$. L'espérance de ces variables aléatoires est alors donnée par le théorème suivant.

Théorème 10 *Pour tout $\upsilon \geq 1$ et pour tout $m \geq 0$, on a*

$$\frac{E(N_S(m))}{\upsilon} = \alpha \left(\frac{1}{\upsilon} T + \left(1 - \frac{1}{\upsilon}\right) I\right)^m \mathbb{1}_S. \tag{7.2}$$

$$\frac{E(N_P(m))}{\upsilon} = \alpha \left(\frac{1}{\upsilon} T + \left(1 - \frac{1}{\upsilon}\right) I\right)^m \mathbb{1}_P. \tag{7.3}$$

Preuve D'après la définition de la variable aléatoire $N_S(m)$ et le théorème 9, on a

$$\begin{aligned} E(N_S(m)) &= \sum_{j=1}^{\upsilon} \Pr\{X_m^{(j)} \in S\} = \sum_{j=1}^{\upsilon} \sum_{(s,x,y) \in S} \Pr\{X_m^{(j)} = (s,x,y)\} \\ &= \upsilon \sum_{\ell=0}^{m} \binom{m}{\ell} \left(\frac{1}{\upsilon}\right)^{\ell} \left(1 - \frac{1}{\upsilon}\right)^{m-\ell} \Pr\{X_\ell \in S\}. \end{aligned}$$

D'autre part, on a $\Pr\{X_\ell \in S\} = \alpha T^\ell \mathbb{1}_S$. On a donc

$$\begin{aligned} E(N_S(m)) &= \upsilon \sum_{j=0}^{m} \binom{m}{j} \left(\frac{1}{\upsilon}\right)^j \left(1-\frac{1}{\upsilon}\right)^{m-j} \alpha T^j \mathbb{1}_S \\ &= \upsilon\alpha \sum_{j=0}^{m} \binom{m}{j} \left(\frac{1}{\upsilon}T\right)^j \left(1-\frac{1}{\upsilon}\right)^{m-\ell} \mathbb{1}_S \\ &= \upsilon\alpha \left(\frac{1}{\upsilon}T + \left(1-\frac{1}{\upsilon}\right) I\right)^m \mathbb{1}_S. \end{aligned}$$

Le calcul de $E(N_P(m))$ s'effectue de manière similaire. □

Les états de $S \cup P$ sont transitoires, par conséquent pour tout $\upsilon \geq 1$, les matrices T et $(1/\upsilon)T + (1-1/\upsilon)I$ sont sous-stochastiques. On a alors pour tout $\upsilon \geq 1$:

$$\lim_{m\longrightarrow\infty} \frac{E(N_S(m))}{\upsilon} = \lim_{m\longrightarrow\infty} \frac{E(N_P(m))}{\upsilon} = 0.$$

D'autre part, $E(N_S(m))$ est croissant avec υ. On a pour tout $m \geq 0$:

$$\lim_{\upsilon\longrightarrow\infty} \left(\frac{1}{\upsilon}T + \left(1-\frac{1}{\upsilon}\right) I\right)^m = I$$

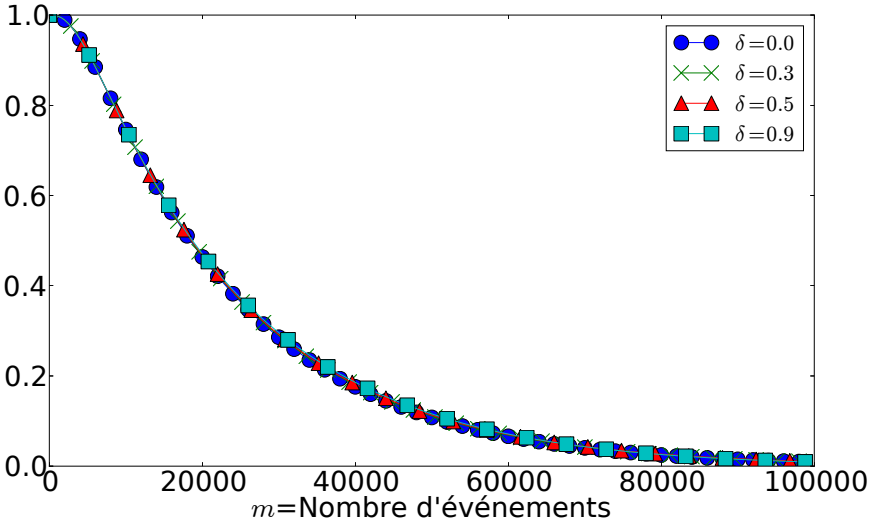
$$\lim_{\upsilon\longrightarrow\infty} E(N_S(m)) = \lim_{\upsilon\longrightarrow\infty} E(N_P(m)) = +\infty.$$

$$\lim_{\upsilon\longrightarrow\infty} \frac{E(N_S(m))}{\upsilon} = \alpha\mathbb{1}_S \quad \text{et} \quad \frac{E(N_P(m))}{\upsilon} = \alpha\mathbb{1}_P.$$

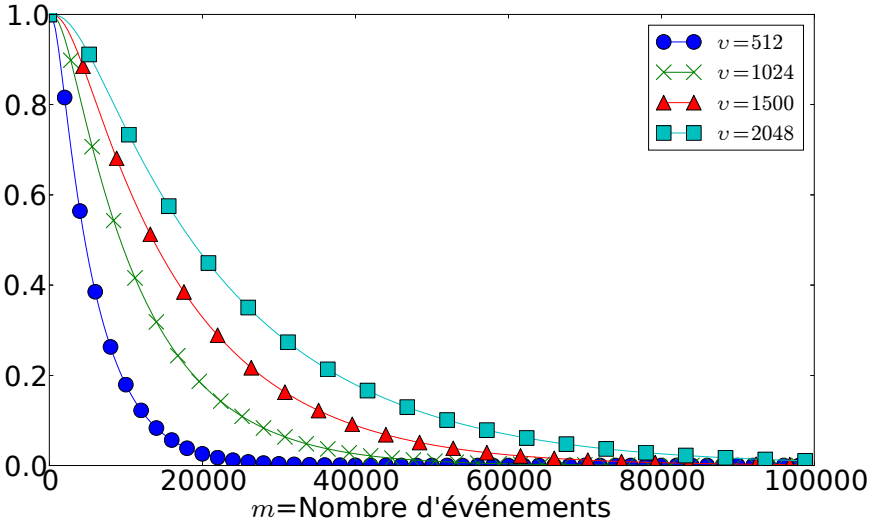
Les figures 7.1 et 7.2 illustrent respectivement les proportions moyennes de clusters sains et pollués en fonction du nombre d'événements `join` et `leave` sur le système pour différents nombres de cluster υ et différentes valeurs de δ représentant la probabilité qu'un identifiant soit valide.

On constate sur la figure 7.1(a) que la probabilité δ qu'un identifiant soit valide n'a qu'une très faible influence sur la proportion de clusters sains dans le système. En effet, bien que la valeur de δ ait plus d'influence sur la proportion de clusters pollués, comme on peut le voir sur la figure 7.2(a), cet impact se limite à 0.1% d'écart entre une probabilité $\delta = 0$ correspondant à une durée de validité des identifiants quasi-nulle et une probabilité δ proche de 1 correspondant à une durée de validité des identifiants quasi-infinie. Ce faible impact s'explique par le fait que le churn naturel domine le churn induit lié aux durées de validité des identifiants. Cela permet donc ainsi de limiter la proportion de clusters pollués sans impacter celle des clusters sains.

D'autre part, on constate que dans tous les cas, la proportion de clusters pollués reste extrêmement faible ($E(N_P(m))/\upsilon \leq 1\%$) démontrant ainsi l'intérêt du churn induit par les identifiants à durée limitée et la résilience de PeerCube face aux comportements byzantins.

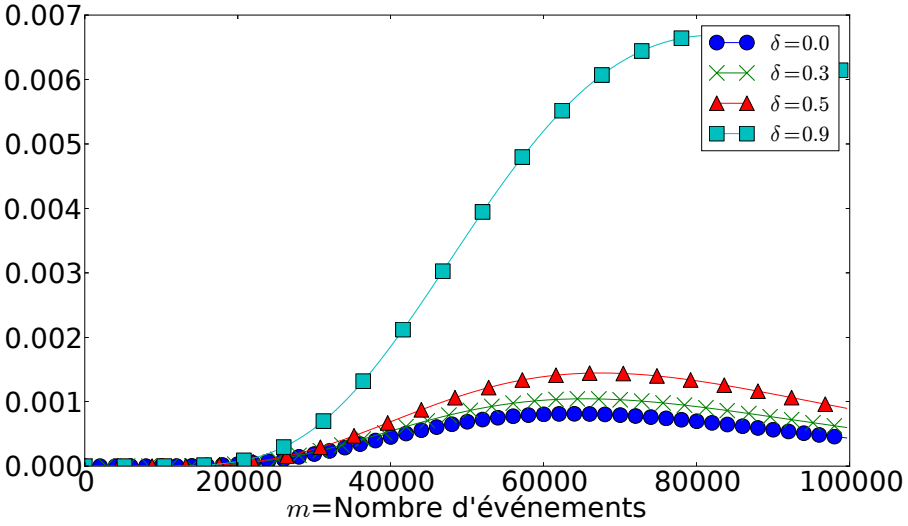


(a) $\frac{E(N_S(m)}{v}$, $v = 2048$, $\mu = 25\%$, $\alpha = \alpha_1$

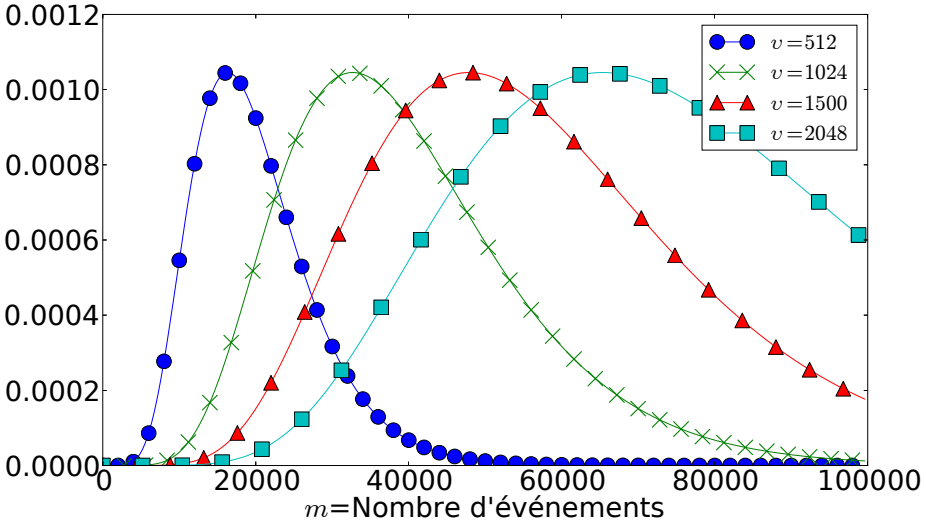


(b) $\frac{E(N_S(m))}{v}$, $\delta = 30\%$, $\mu = 25\%$, $\alpha = \alpha_1$

FIGURE 7.1 – $E(N_S(m))/v$ dans le cas de P_1 en fonction du nombre m d'événements `join` et `leave` pour $\alpha = \alpha^{(1)}$ et différents nombres de cluster v et différentes probabilités δ de validité des identifiants.



(a) $\frac{E(N_P(m)}{\upsilon}$, $\upsilon = 2048$, $\mu = 25\%$, $\alpha = \alpha_1$



(b) $\frac{E(N_P(m))}{\upsilon}$, $\delta = 30\%$, $\mu = 25\%$, $\alpha = \alpha_1$

FIGURE 7.2 – $E(N_P(m))/\upsilon$ dans le cas de P_1 en fonction du nombre m d'événements `join` et `leave` pour $\alpha = \alpha^{(1)}$ et différents nombres de cluster υ et différentes probabilités δ de validité des identifiants.

7.1.2 Instant du premier changement topologique

Nous venons de calculer l'espérance de la proportion de clusters dans les états sains et pollués dans le système. Du point de vue de notre modélisation, cela se traduit par la proportion moyenne de chaînes de Markov $X^{(j)}, 1 \leq j \leq \upsilon$ appartenant aux états de S ou P.

Les chaînes étant absorbantes, ces proportions tendent vers 0. Cette mesure est donc une sur-approximation du nombre moyens de clusters sains et pollués dans le système. En effet, dès qu'un cluster doit se scinder ou fusionner, le nombre total de clusters change et donc la probabilité qu'ils reçoivent un événement de `join` et `leave` varie.

On cherche alors à déterminer la loi de ces proportions avant le premier instant de changement topologique dans le système. Un changement topologique s'opère dans le système dans deux cas :

— un cluster du système devient surpeuplé et celui-ci doit se scinder en deux nouveaux clusters (opération `split`),

— un cluster devient sous-peuplé et doit donc fusionner avec un autre cluster du système (opération `merge`).

On définit $p_\psi(m)$ comme la probabilité que ψ clusters parmi υ soient dans un état pollué et aucun des υ clusters n'ait déclenché d'opération `split` ou `merge`. Cela signifie donc de manière plus formelle que pour $\psi = 0, \ldots, \upsilon$ on a ψ chaînes de Markov dans un état pollué et $\upsilon - \psi$ dans un état sain. On peut donc écrire formellement :

$$\forall \psi \in [\![0, \upsilon]\!], \quad p_\psi(m) = \Pr\{N_S(m) = \upsilon - \psi, N_P(m) = \psi\}.$$

On définit pour tout $m \geq 0$ et tout $1 \leq \ell \leq n$, l'ensemble $S_{m,\ell}$ par :

$$S_{m,\ell} = \{\underline{m} = (m_1, \ldots, m_\ell) \in \mathbb{N}^\ell \mid m_1 + \cdots + m_\ell = m\}.$$

Nous introduisons les notations suivantes :

$$q_S(k) = \Pr\{X_k \in S\} = \alpha_T T^k \mathbb{1}_S \quad \text{et} \quad q_P(k) = \Pr\{X_k \in P\} = \alpha_T T^k \mathbb{1}_P. \tag{7.4}$$

La probabilité $p_\psi(m)$ a été étudiée dans [ALS12]. On a pour tout $m \geq 0$ et pour tout $\psi = 0, \ldots, \upsilon$:

$$p_\psi(m) = \binom{\upsilon}{\psi} \frac{1}{\upsilon^m} \sum_{\underline{m} \in S_{m,\upsilon}} \frac{m!}{m_1! \cdots m_\upsilon!} \prod_{j=1}^{\psi} q_P(m_j) \prod_{j=\psi+1}^{\upsilon} q_S(m_j). \tag{7.5}$$

Pour $\psi = 0$ (respectivement $\psi = \upsilon$), le produit $\displaystyle\prod_{j=1}^{\psi} q_P(m_j)$ (respectivement $\displaystyle\prod_{j=\psi+1}^{\upsilon} q_S(m_j)$) est vide. On considère alors par convention que le produit vide vaut 1.

Soit Θ_υ le premier instant auquel l'une des υ chaînes de Markov $X^{(1)}, \ldots, X^{(\upsilon)}$ entre dans une classe absorbante. Cette variable aléatoire est définie par :

$$\Theta_\upsilon = \inf\{m \geq 0 \mid \exists j \in [\![1, \upsilon]\!] \text{ tel que } X_m^{(j)} \in A_S^\omega \cup A_S^\sigma \cup A_P^\omega\}.$$

Cette variable aléatoire a été étudiée dans [ACLS13]. On a pour tout $m \geq 0$ et et pour tout $\nu = 1, \ldots, n-1$,

$$\Pr\{\Theta_\upsilon > m\} = \sum_{j=0}^{m} \binom{m}{\ell} \left(\frac{\nu}{\upsilon}\right)^j \left(1 - \frac{\nu}{\upsilon}\right)^{m-j} \Pr\{\Theta_\nu > j\} \Pr\{\Theta_{\upsilon-\nu} > m - j\}. \quad (7.6)$$

D'autre part, pour tout $m \geq 0$, on a :

$$\lim_{\upsilon \longrightarrow \infty} \Pr\{\Theta_\upsilon > m\} = (\alpha_T T \, \mathbb{1})^m. \quad (7.7)$$

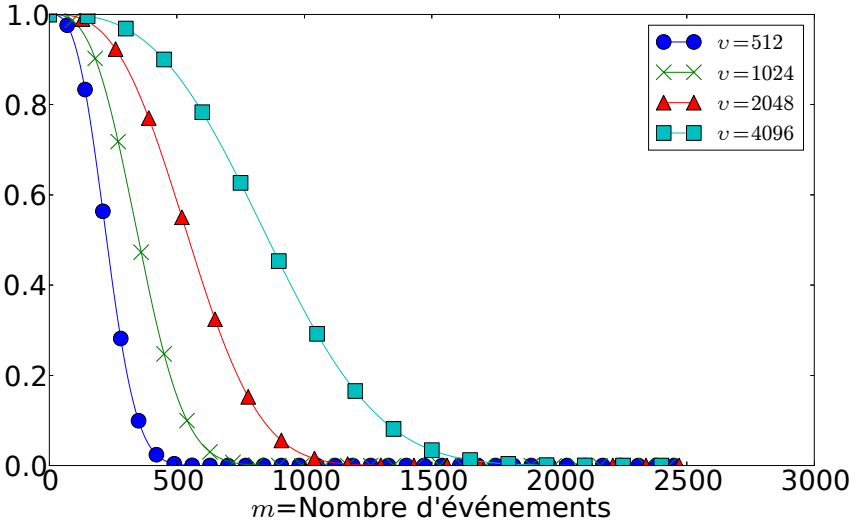
Les figures 7.3 et 7.4 illustrent respectivement la loi du premier instant auquel l'une des υ chaînes de Markov entre dans une classe d'états absorbante en fonction du nombre m d'événements `join` et `leave` dans le système, pour différents nombres de clusters υ (figure 7.3) et différentes valeurs de δ (figure 7.4) représentant la probabilité qu'un identifiant soit valide. Enfin, ces distributions sont calculées pour $\gamma = 7$ et $\Gamma = 14$ représentant ainsi un ensemble d'entités supervisées dans PeerCube de taille h comprise entre $7168 \leq h \leq 57344$ entités.

On constate sur les figures 7.3(a) et 7.3(b) que plus le nombre de clusters augmente, plus le nombre d'événements nécessaires pour qu'une chaîne entre dans une classe d'états absorbante augmente. En effet, les événements étant uniformément distribués sur l'ensemble des clusters, plus cet ensemble grandit, plus la probabilité qu'un cluster particulier soit choisi pour effectuer une transition diminue, repoussant ainsi l'instant du premier changement topologique.

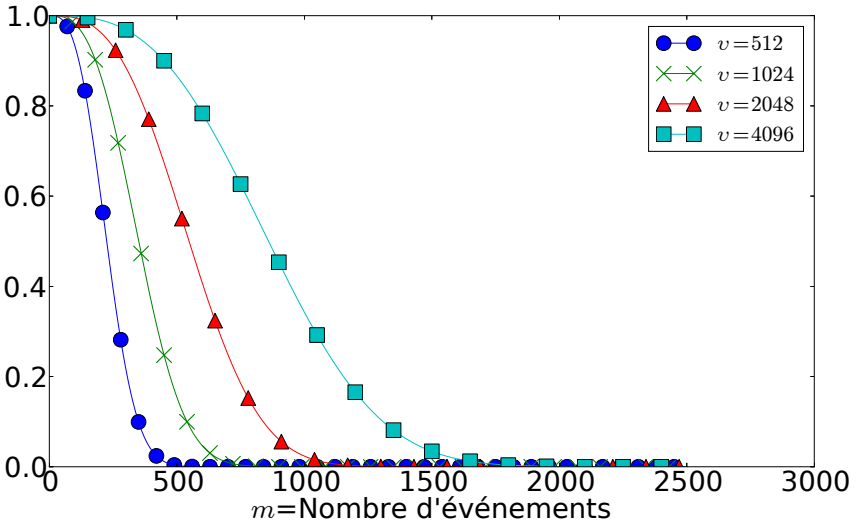
D'autre part, on constate sur la figure 7.4(a) que la durée de validité des identifiants n'a pas d'influence sur l'instant du premier changement topologique. En effet, dans ce cas, les clusters sont initialement exempts d'entité malveillantes, c'est-à-dire que chacune des chaînes de Markov $X^{(j)}, 1 \leq j \leq \upsilon$ représentant la composition du cluster $\mathcal{C}_j$ est initialisée avec la distribution $\alpha^{(1)}$. À l'inverse, dans le cas de la figure 7.4(b), les clusters comportent initialement des entités malveillantes. Les états initiaux des chaînes de Markov $X^{(j)}, 1 \leq j \leq \upsilon$ sont distribués selon $\alpha^{(2)}$. Dans ce cas, la loi $\Pr\{\Theta_\upsilon > m\}$ du premier changement topologique croît avec la valeur de δ, représentant la probabilité qu'un identifiant soit valide.

Dans le premier cas, le nombre d'opérations au niveau d'un cluster permettant d'atteindre un état appartenant à une classe d'états absorbante est inférieur au temps nécessaire à l'adversaire pour polluer le cluster et ainsi ignorer certains événements `join` et `leave`. Dans le second cas, des entités malveillantes sont déjà présentes dans le cluster réduisant ainsi ce temps et permettant alors à l'adversaire d'ignorer ces événements. La durée de validité des identifiants a alors une influence sur le temps nécessaire au cluster pour atteindre une classe d'états absorbante et accroît $\Pr\{\Theta_\upsilon > m\}$.

Comme nous l'avons vu auparavant, la quantité $E(N_S(m))$ est une sur-approximation du nombre moyen de clusters sains présents dans le système. En effet, dès qu'un cluster doit se scinder ou fusionner, le nombre total de clusters change et donc la probabilité qu'ils reçoivent un événement de `join` et `leave` varie. Par conséquent, on cherche à déterminer la proportion moyenne de clusters sains avant le premier changement topologique dans le système.

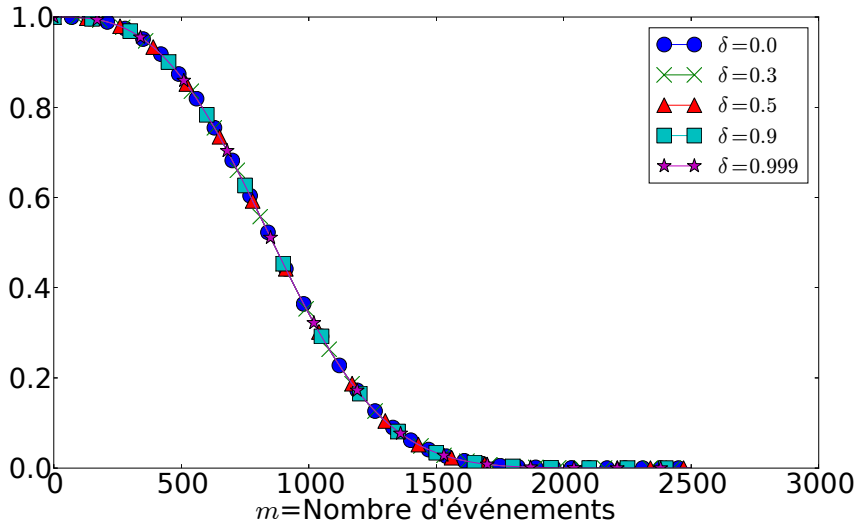


(a) $\delta = 30\%$ et $\alpha = \alpha^{(1)}$

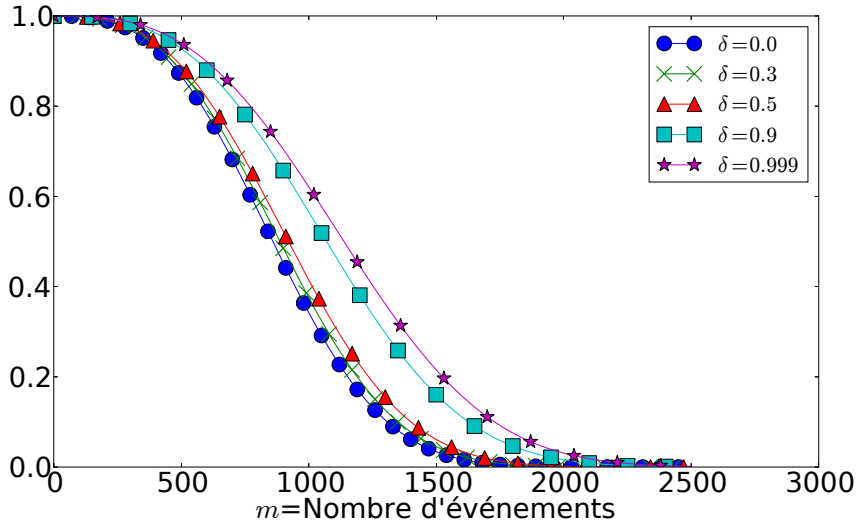


(b) $\delta = 90\%$ et $\alpha = \alpha^{(1)}$

FIGURE 7.3 – $\Pr\{\Theta_n > m\}$ (Relation (7.6)) en fonction de m, v pour $\mathcal{P}_{\phi=1}$ dans le cas où $\alpha = \alpha^{(1)}$, $\gamma = 7$, $\Gamma = 14$, et $\mu = 25\%$.



(a) $\upsilon = 4096$ et $\alpha = \alpha^{(1)}$



(b) $\upsilon = 4096$ et $\alpha = \alpha^{(2)}$

FIGURE 7.4 – $\Pr\{\Theta_n > m\}$ (Relation (7.6)) en fonction de m, δ pour les deux distributions initiales $\alpha^{(1)}$ et $\alpha^{(2)}$ pour $\mathcal{P}_{\phi=1}$ dans le cas où $\upsilon = 4096$, $\gamma = 7$, $\Gamma = 14$, et $\mu = 25\%$.

La quantité $E(N_S(m)1_{\{\Theta_\upsilon > m\}})$ correspond alors au nombre moyen de chaînes de Markov dans une classe d'états sains à l'instant m avant qu'une de ces chaînes entre dans une classe d'états absorbante. De la même manière, la quantité $E(N_P(m)1_{\{\Theta_\upsilon > m\}})$ correspond au nombre moyen de chaînes de Markov dans une classe d'états pollués à l'instant m avant qu'une de ces chaînes entre dans une classe d'états absorbante.

Pour tout $\psi = 0, \dots \upsilon$, on a l'équivalence suivante :

$$(N_S(m) = \psi, N_P(m) = \upsilon - \psi) \Leftrightarrow (N_S(m) = \psi, \Theta_\upsilon > m).$$

On a montré dans [ALS12] qu'on peut calculer ces quantités de la manière suivante. Pour tout $m \geq 0$, on a :

$$E(N_S(m)1_{\{\Theta_\upsilon > m\}}) = \upsilon \sum_{j=0}^{m} \binom{m}{j} \left(\frac{1}{\upsilon}\right)^j \left(1 - \frac{1}{\upsilon}\right)^{m-j} q_S(j) \Pr\{\Theta_{\upsilon-1} > m - j\}, \quad (7.8)$$

$$E(N_P(m)1_{\{\Theta_\upsilon > m\}}) = \upsilon \sum_{j=0}^{m} \binom{m}{j} \left(\frac{1}{\upsilon}\right)^j \left(1 - \frac{1}{\upsilon}\right)^{m-j} q_P(j) \Pr\{\Theta_{\upsilon-1} > m - j\}. \quad (7.9)$$

On a donc :

$$\frac{E(N_S(m)1_{\{\Theta_\upsilon > m\}})}{\upsilon} + \frac{E(N_P(m)1_{\{\Theta_\upsilon > m\}})}{\upsilon} = \Pr\{\Theta_\upsilon > m\}.$$

D'autre part, d'après la relation (7.7) on a :

$$\lim_{\upsilon \longrightarrow \infty} \frac{E(N_S(m)1_{\{\Theta_\upsilon > m\}})}{\upsilon} = q_S(0)(\alpha_T T \mathbb{1})^m = \alpha_S \mathbb{1}(\alpha_T T \mathbb{1})^m,$$
$$\lim_{\upsilon \longrightarrow \infty} \frac{E(N_P(m)1_{\{\Theta_\upsilon > m\}})}{\upsilon} = q_P(0)(\alpha_T T \mathbb{1})^m = \alpha_P \mathbb{1}(\alpha_T T \mathbb{1})^m.$$

Les figures 7.5 et 7.6 illustrent l'évolution des deux quantités $E(N_S(m)1_{\{\Theta_n > m\}})$ et $E(N_P(m)1_{\{\Theta_n > m\}})$ pour un système composé de $\upsilon = 4097$ clusters pour différentes probabilités δ de validité des identifiants. Les figures 7.5(a) et 7.6(a) représentent ces quantités pour une distribution initiale des états $\alpha^{(1)}$ et les figures 7.5(b) et 7.6(b) dans le cas de $\alpha = \alpha^{(2)}$.

On constate alors que lorsque les clusters sont initialisés avec $\alpha^{(1)}$ la proportion de clusters en état sain avant le premier changement topologique est très importante tandis que la proportion de clusters en état pollué est négligeable. Ce résultat confirme que les procédures de renouvellement du core sont robustes à la présence de l'adversaire.

En effet, le churn naturel domine ici l'adversaire et les clusters se scindent ou fusionnent avant même que ce dernier réussisse à polluer le cluster. D'autre part, on constate que les clusters ont une durée de vie plus importante en présence d'entités malveillantes. En raison de la règle 1, l'adversaire ne retire ses entités malveillantes qu'à l'échéance de leurs identifiants. Ainsi, certains événements `leave` sont ignorés augmentant alors la durée de vie du cluster.

Nous avons montré dans cette section qu'en combinant des procédures de renouvellement robustes et en introduisant un churn induit au niveau des entités du système, celui-ci se montre résilient à la présence d'un adversaire visant à corrompre le système.

De manière contre intuitive, le protocole $P_{\phi=1}$ choisissant une unique entité dans le spare se montre plus efficace que de renouveler l'ensemble du core que les autres protocoles $P_{\phi>1}$. De cette manière, une seule instance de consensus byzantin est nécessaire pour déterminer l'entité du spare choisie pour intégrer le core. À l'inverse, dans le cas des protocoles $P_{\phi>1}$, une première instance de consensus byzantin est nécessaire pour déterminer les $\phi - 1$ entités du core à placer dans le spare, avant de sélectionner les ϕ entités à inclure dans le nouveau core. Par conséquent, le protocole $P_{\phi=1}$ se montre moins coûteux d'un point de vue algorithmique que les autres protocoles $P_{\phi>1}$.

D'autre part, en introduisant des durées de validité pour les identifiants, l'adversaire ne peut laisser ses entités malveillantes en place pour polluer progressivement le système. Nous avons montré qu'avec ces durées de validité, l'adversaire est contraint et ne réussi à polluer qu'une faible partie des clusters du système. De plus, les clusters pollués le sont temporairement et cette pollution ne se propage pas dans le système. Enfin, nous avons montré que le churn induit par l'introduction de ces durées de validités peut être limité tout en limitant la capacité de pollution de l'adversaire.

7.2 Routage

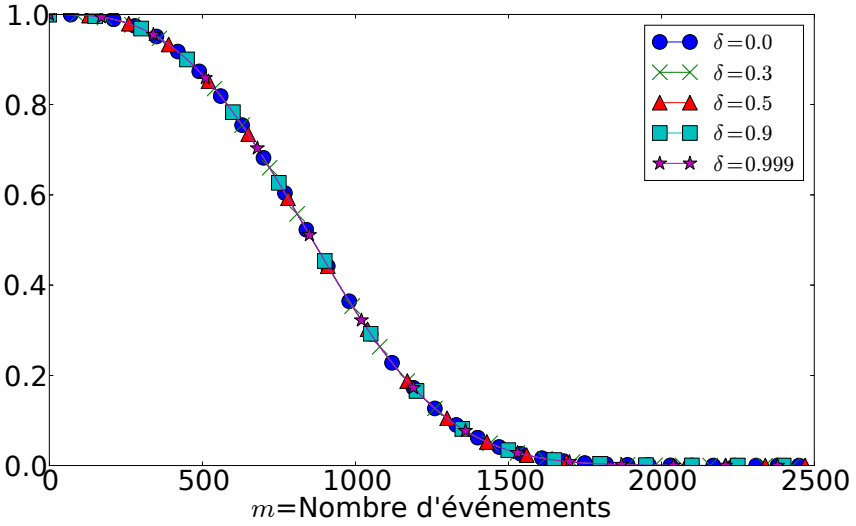
Nous avons montré dans les sections précédentes la robustesse du protocole vis à vis de la procédure de renouvellement ainsi que vis à vis du churn induit par les durées de validité des identifiants. Malgré cette robustesse nous avons montré qu'une très faible proportion de clusters pouvaient être maîtrisés par l'adversaire. Dans cette section, nous quantifions l'impact de la présence de clusters pollués sur la procédure de `lookup` présentée au chapitre précédent puis nous présentons une autre procédure de `lookup` avant d'évaluer sa résilience à l'adversaire.

Les entités de PeerCube dont les identifiants partagent un préfixe commun sont regroupées en clusters de sorte que la propriété 5 soit vérifiée. Ce préfixe est appelé label du cluster. Les clusters de PeerCube sont organisés suivant une topologie d'hypercube. Deux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$ de labels respectifs L_1, L_2 sont voisins dans cet hypercube si la distance de Hamming entre leurs labels respectifs vaut 1.

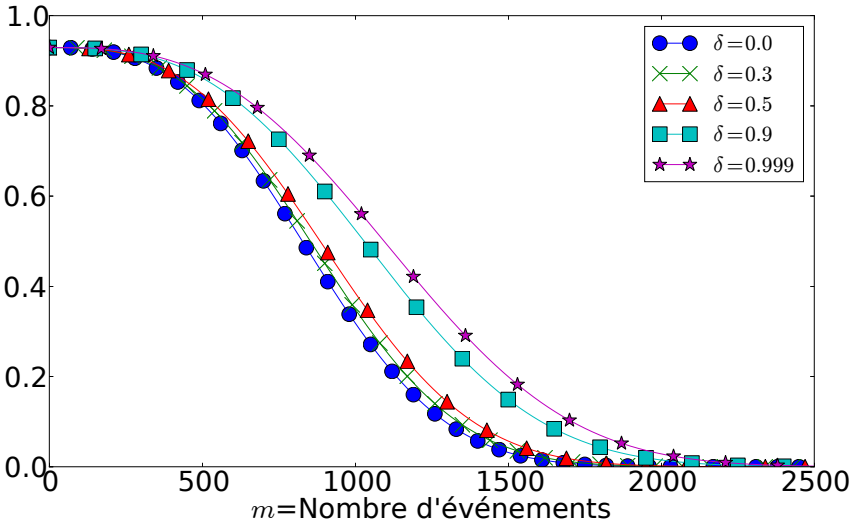
La procédure classique de `lookup` sur un hypercube allant d'un sommet labellisé L_1 à un sommet labellisé L_2 consiste à modifier successivement les bits de L_1 qui diffèrent de L_2 pour obtenir L_2.

Exemple 30 *La figure 7.7 illustre le déroulement de l'opération* **`lookup`** *sur la topologie précédente. On considère qu'une entité du cluster labellisé "011" souhaite effectuer une opération* **`lookup`** *sur un indentifiant préfixé par "110". La distance de hamming entre ces deux labels vaut* 2*, les clusters correspondants en sont donc pas voisins dans l'hypercube.*

Les entités du core du cluster labellisé "011" modifient le premier bit du label du cluster qui diffère du label cible. On obtient alors "111". Une requête de **`lookup`** *est envoyée au cluster labellisé "111". Le core de ce cluster modifie le premier bit de "111" qui diffère du label cible et obtient "110". Les entités du core font alors suivre la reqête de* **`lookup`** *au core du cluster "110". La requête est arrivée à destination.*

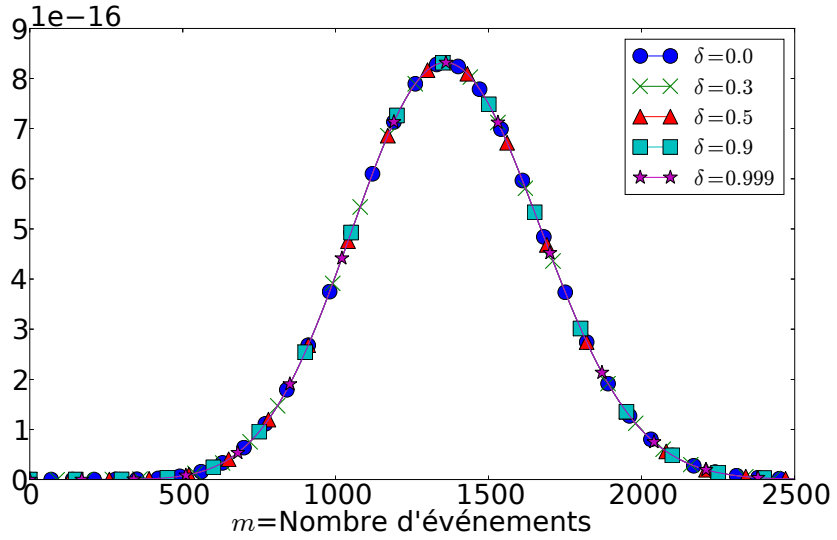


(a) $\alpha = \alpha^{(1)}$

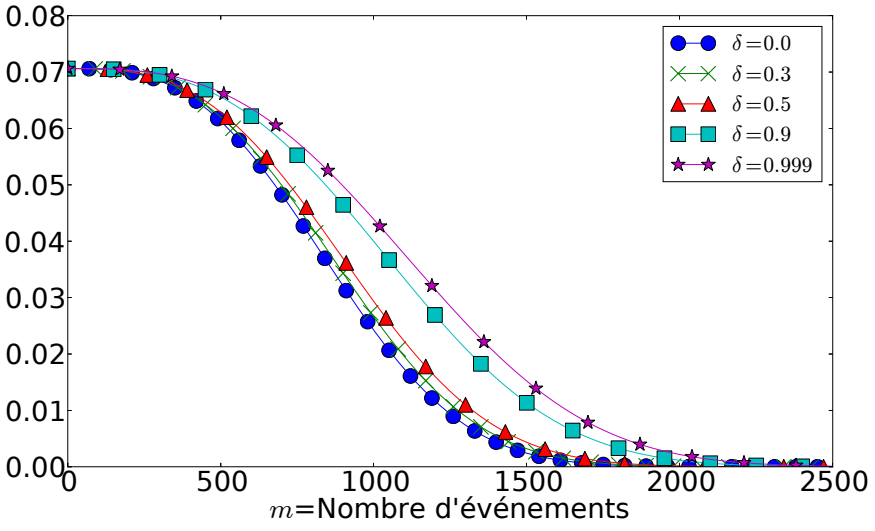


(b) $\alpha = \alpha^{(2)}$

FIGURE 7.5 – $E(N_S(m)1_{\{\Theta_v > m\}})/n$ en fonction de m et δ pour P_1 dans le cas où $v = 4097$ clusters, $\gamma = 7$, $\Gamma = 14$, $\eta = 0.2$, et $\mu = 25\%$.



(a) $\alpha = \alpha^{(1)}$



(b) $\alpha = \alpha^{(2)}$

FIGURE 7.6 – $E(N_P(m)1_{\{\Theta_\upsilon > m\}})/n$ en fonction de m et δ pour P_1 dans le cas où $\upsilon = 4097$ clusters, $\gamma = 7$, $\Gamma = 14$, $\eta = 0.2$, et $\mu = 25\%$.

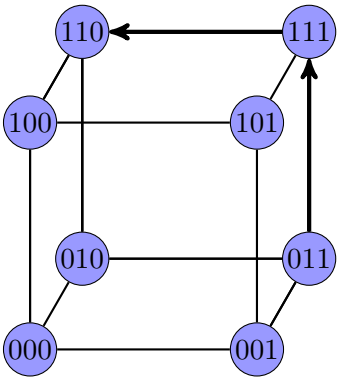
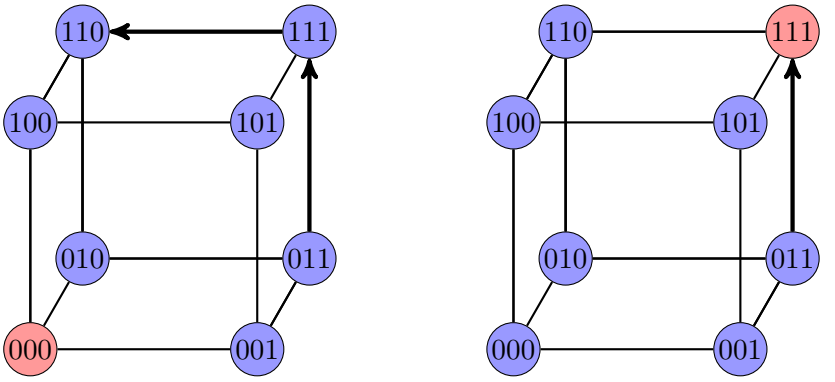


FIGURE 7.7 – Routage d'une requête `lookup` dans PeerCube du cluster "011" au cluster "110".



(a) Succès d'une requête `lookup` (b) Échec d'une requête `lookup`

FIGURE 7.8 – Routage d'une requête `lookup` dans PeerCube.

Comme nous l'avons vu au chapitre précédent, l'adversaire peut polluer certains clusters du système. Dans ce cas, il a tout contrôle sur les messages émis et reçus par ce cluster. Par conséquent, il peut décider de bloquer une procédure de `lookup`.

Exemple 31 *Considérons la situation illustrée dans la figure 7.8. Cette figure décrit le chemin enprunté par une requête* `lookup` *allant du cluster "011" au cluster "110" en présence de clusters pollués.*

Dans le premier cas illustré dans la figure 7.8(a), le cluster labellisé "000" est pollué (représenté par un diamant rouge), tandis que les autres clusters sont sains. Le cluster "000" n'est donc pas sur le chemin de la requête de `lookup`*, celle-ci arrive donc au cluster "011" et la requête est satisfaite. À l'inverse, dans le cas illustré dans la figure 7.8(b), le cluster labellisé "111" est pollué. Ce cluster est situé sur le chemin emprunté par la requête de* `lookup`*. Le cluster pollué bloque la requête, celle-ci échoue.*

On considère un système composé de v clusters. Ce système subit des arrivées et des départs d'entités et on se place avant le premier changement topologique. On considère

une requête de `lookup` effectuée par une entité du cluster $\mathcal{C}_1$ à destination du cluster $\mathcal{C}_\ell$. Le chemin emprunté par la requête est noté $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell)$. La composition de chaque cluster $\mathcal{C}_j, 1 \leq j \leq \ell$ est représentée par la chaîne de Markov $X^{(j)}$ identique à X. Les autres clusters du système sont représentés par les chaînes $X^{(\ell+1)}, \ldots, X^{(\upsilon)}$. On considère qu'une recherche `lookup` aboutit si le chemin emprunté par celle-ci ne comporte pas de cluster pollué.

On définit $L(\upsilon, m, (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell))$ comme la probabilité de succès du requête de `lookup` effectuée le long du chemin $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell)$ dans ce système. Les chaînes $X^{(j)}$ étant identiques, la probabilité $L(n, m, (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell))$ ne dépend que de la longueur du chemin $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell)$ emprunté par la requête de `lookup` et non de l'identité des ℓ clusters qui le composent. On simplifie alors la définition de cette probabilité :

$$L(\upsilon, m, \ell) = L(\upsilon, m, (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell))$$

avec $L(\upsilon, m, \ell)$ la probabilité de succès d'une requête `lookup` le long d'un chemin de longueur ℓ dans un système composé de υ clusters après la m-ème opération de `join` ou `leave` dans le système et avant le premier instant de changement topologique.

Par définition, on a :

$$\begin{aligned} L(\upsilon, m, \ell) &= \Pr\{X_m^{(1)} \in S, \ldots, X_m^{(\ell)} \in S, \Theta_\upsilon > m\} \\ &= \Pr\{X_m^{(1)} \in S, \ldots, X_m^{(\ell)} \in S, X_m^{(\ell+1)} \in S \cup P, \ldots, X_m^{(\upsilon)} \in S \cup P\} \end{aligned}$$

De manière similaire à la relation (7.5), on a :

$$L(\upsilon, m, \ell) = \frac{1}{\upsilon^m} \sum_{\underline{m} \in S_{m,n}} \frac{m!}{m_1! \cdots m_n!} \prod_{r=1}^{\ell} q_S(m_r) \prod_{r=\ell+1}^{\upsilon} q_{S \cup P}(m_r),$$

avec $q_{S \cup P}(m_r) = \Pr\{X_{m_r}^{(r)} \in S \cup P\}$.
On a alors la récurrence suivante

$$\begin{aligned} L(\upsilon, m, \ell) &= \frac{1}{\upsilon^m} \sum_{\underline{m} \in S_{m,\upsilon}} \frac{m!}{m_1! \cdots m_\upsilon!} q_S(m_\upsilon) \prod_{j=1}^{\upsilon-\ell} q_{S \cup P}(m_j) \prod_{j=\upsilon-\ell+1}^{\upsilon-1} q_S(m_j) \\ &= \frac{1}{\upsilon^m} \sum_{j=0}^{m} \binom{m}{j} q_S(j) (\upsilon - 1)^{m-j} L(\upsilon - 1, m - j, \ell - 1) \\ &= \sum_{j=0}^{m} \binom{m}{j} \left(\frac{1}{\upsilon}\right)^j \left(1 - \frac{1}{\upsilon}\right)^{m-j} q_S(j) L(\upsilon - 1, m - j, \ell - 1). \end{aligned}$$

avec pour conditions initiales : $\forall k = 0, \ldots, m,\ L(n - \ell, k, 0) = \Pr\{\Theta_{n-\ell} > k\}$.

Les clusters de PeerCube sont connectés entre eux de manière à former un hypercube. Deux clusters $\mathcal{C}_1$ et $\mathcal{C}_2$ sont voisins si la distance de Hamming entre leurs labels L_1, L_2 respectifs vaut 1 :

$$\mathcal{C}_1 \text{ et } \mathcal{C}_2 \text{ voisins} \iff \texttt{Hamming}(L_1, L_2) = 1$$

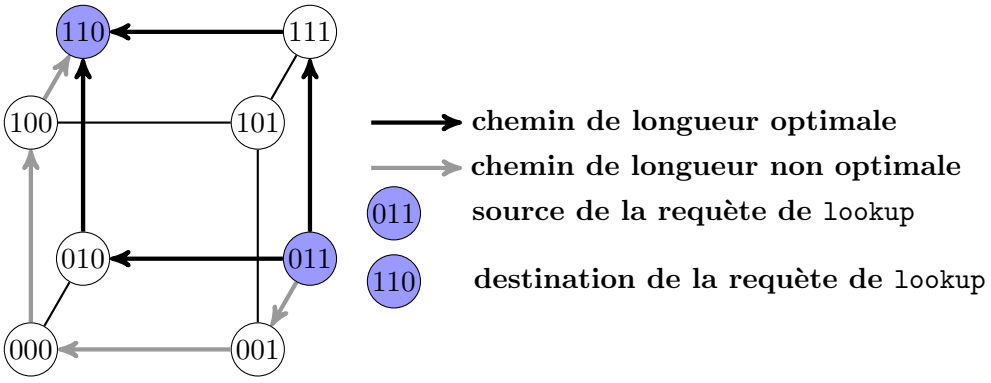


FIGURE 7.9 – Chemins indépendants de “011” à “110”.

Il existe différents chemins reliant une cluster $\mathcal{C}_1$ à un cluster $\mathcal{C}_2$. Deux chemins sont dits indépendants s'ils ne partagent que la source et la destination. Le nombre de chemins indépendants entre $\mathcal{C}_1$ et $\mathcal{C}_2$ est donné par la propriété 7 dont la preuve se situe dans [SS88].

Propriété 7 (Chemins indépendants) *Soient $\mathcal{C}_1$ et $\mathcal{C}_2$ deux clusters de PeerCube de labels respectifs L_1 et L_2. Ces clusters sont deux des sommets d'un hypercube de dimension ℓ engendré par PeerCube. Il existe alors ℓ chemins indépendants entre $\mathcal{C}_1$ et $\mathcal{C}_2$ et leurs longueur est inférieure à $\texttt{Hamming}(L_1, L_2) + 2$.*

PeerCube utilise ces chemins indépendants afin d'effectuer une recherche `lookup` sur chacun de ces chemins en parallèle. Contrairement au `lookup` classique qui modifie successivement les bits du label du sommet v pour obtenir le label du sommet w en commençant par le premier bit qui diffère entre v et w, la modification pour le i-ème chemin redondant commence au i-ème bit qui diffère entre v et w.

Exemple 32 *La figure 7.9 illustre les chemins indépendants entre le cluster labellisé ”011” et le cluster labellisé ”110” dans un hypercube de dimension 3.*

On a $\texttt{Hamming}(\text{”011”}, \text{”110”}) = 2 \neq 1$, les clusters ”011” et ”110” ne sont pas voisins. D'après la propriété 7, il existe 2 chemins indépendants de longueur optimale 2 et un 3-ème chemin dont la longueur n'excède pas $\texttt{Hamming}(\text{”011”}, \text{”110”}) + 2 = 4$.

- *Le premier chemin de longueur optimale est obtenu en modifiant le premier bit de ”011” qui diffère de ”110”. On identifie chaque cluster par son label. On construit alors le chemin* $(011, 111, 110)$.
- *Le second chemin de longueur optimale est obtenu en modifiant le second bit de ”011” qui diffère de ”110”. On construit alors le chemin* $(011, 010, 110)$.
- *Enfin, on construit le troisième chemin indépendant en modifiant le premier bit de ”011” commun à ”110”. On construit alors le chemin* $(011, 001, 000, 100, 110)$ *de longueur 4.*

La possibilité d'emprunter des chemin indépendants permet ainsi de tolérer la présence de clusters pollués. En effet, la probabilité de rencontrer un cluster pollué

décroît exponentiellement avec le nombre de chemins indépendants [SL04]. De plus, l'utilisation de chemin de longueur optimale diminue la probabilité d'occurrence d'un cluster pollué sur ce chemin.

On considère un système composé de $\upsilon = 2^\ell$ clusters formant un hypercube parfait. Ce système subit des arrivées et des départs d'entités et on se place avant le premier changement topologique. La requête emprunte $l \leq \ell$ chemins indépendants de longueur ℓ. La requête atteint effectivement sa destination s'il existe au moins un chemin parmi les l empruntés dans lequel tous les clusters sont sains.

On note $R(\upsilon, m, l, \ell)$ la probabilité de succès d'une requête de `lookup` dans le système utilisant les chemins redondants immédiatement après le m-ième événement `join` ou `leave` et avant le premier changement topologique.

On désigne par $X^{(1)}$ la chaîne de Markov modélisant la composition du cluster source et par $X^{(\upsilon)}$ celle modélisant la composition du cluster de destination de la requête de `lookup`. De plus, on note $X^{((i-1)(\ell-2)+j)}$ la chaîne de Markov modélisant la composition du j-ième cluster situé sur le chemin i, avec $1 \leq i \leq l$ et $2 \leq j \leq \ell - 1$. La figure 7.10 illustre cette modélisation.

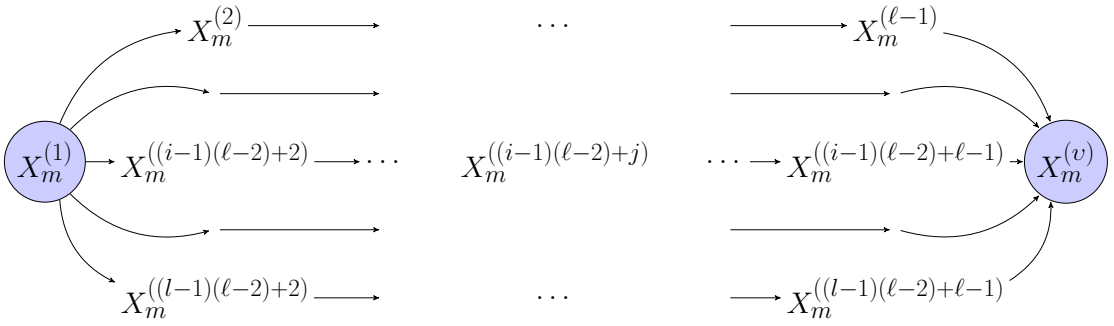


FIGURE 7.10 – Modélisation du routage redondant sur l chemins indépendants de longueur ℓ.

On note $R_m^{(1)}$ l'événement "les ℓ clusters du chemin $i = 1$ sont sains à l'instant m". On a alors :

$$R_m^{(1)} = \{X_m^{(1)} \in S, X_m^{(2)} \in S, \ldots, X_m^{(\ell-1)} \in S, X_m^{(\upsilon)} \in S, \Theta_\upsilon > m\}.$$

De manière plus générale, on note $R_m^{(i)}$ l'événement "les ℓ clusters du chemin i sont sains à l'instant m".

$$R_m^{(i)} = \{X_m^{(1)} \in S, X_m^{((i-1)(\ell-2)+2} \in S, \ldots, X_m^{(i(\ell-2)+1)} \in S, X_m^{(\upsilon)} \in S, \Theta_\upsilon > m\}.$$

La probabilité $R(n, m, l, \ell)$ est alors calculée ainsi :

$$R(\upsilon, m, l, \ell) = \Pr\{\bigcup_{i=1}^{l} R_m^{(i)}\}.$$

D'après le principe d'inclusion-exclusion, on a alors :

$$R(\upsilon, m, l, \ell) = \sum_{j=1}^{l} (-1)^{j+1} \left(\sum_{1 \le i_1 < \cdots < i_j \le j} \Pr\{R_m^{(i_1)}, \ldots, R_m^{(i_j)}\} \right) \tag{7.10}$$

Par définition de $R_m^{(i)}$, on a :

$$\begin{aligned} \Pr\{R_m^{(i_1)}\} &= L(\upsilon, m, \ell), 1 \le i_1 \le j \\ \Pr\{R_m^{(i_1)}, R_m^{(i_2)}\} &= L(\upsilon, m, 2(\ell-2)+2)\ , 1 \le i_1 < i_2 \le j \\ &\cdots \\ \Pr\{R_m^{(1)}, \ldots, R_m^{(j)}\} &= L(\upsilon, m, j(\ell-2)+2) \end{aligned}$$

La relation (7.10) nous donne :

$$j = 1, \sum_{1 \le i_1 < \cdots < i_j \le j} \Pr\{R_m^{(i_1)}, \ldots, R_m^{(i_j)}\} = \Pr\{R_m^{(1)}\} + \ldots + \Pr\{R_m^{(j)}\} = \binom{l}{1} L(\upsilon, m, \ell+2)$$

$$\begin{aligned} j = 2, &\sum_{1 \le i_1 < \cdots < i_j \le j} \Pr\{R_m^{(i_1)}, \ldots, R_m^{(i_j)}\} \\ &= \Pr\{R_m^{(1)}\} + \ldots + \Pr\{R_m^{(j)}\} = \binom{l}{2} L(\upsilon, m, 2(\ell-2)+2) \end{aligned}$$

$$\cdots\cdots$$

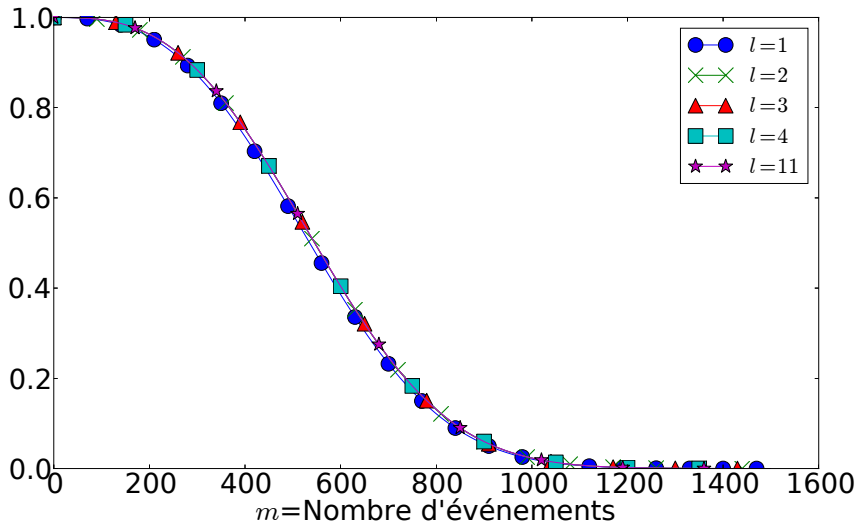
$$\begin{aligned} j = l, &\sum_{1 \le i_1 < \cdots < i_j \le j} \Pr\{R_m^{(i_1)}, \ldots, R_m^{(i_j)}\} \\ &= \binom{l}{l} L(\upsilon, m, l(\ell-2)+2). \end{aligned}$$

On a donc

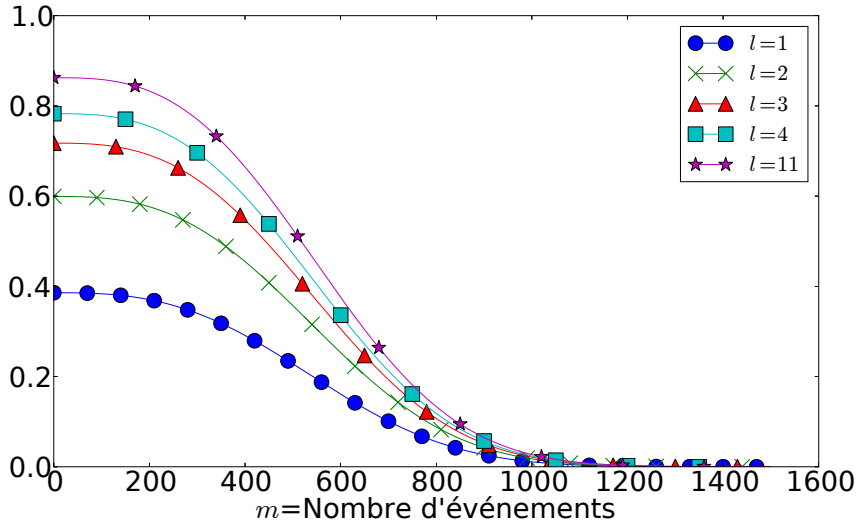
$$\begin{aligned} R(\upsilon, m, l, \ell) =& lL(\upsilon, m, \ell+2) - \binom{l}{2} L(\upsilon, m, 2(\ell-2)+2) + \ldots \\ &+ (-1)^{l-1} L(\upsilon, m, l(\ell-2)+2) \\ =& \sum_{j=1}^{l} \binom{l}{j} L(\upsilon, m, j(\ell-2)+2). \end{aligned}$$

La figure 7.11 compare la probabilité de succès d'une requête de `lookup` simple à celle d'une requête de `lookup` utilisant l chemins redondants dans un système composé de $n = 2^{11} = 2048$ clusters. La distribution initiale des états a un impact très fort sur les valeurs de ces probabilités.

En effet, dans le cas d'un système initialement exempt d'entités malveillantes (voir la figure 7.11(a)), $\alpha = \alpha^{(1)}$), ces probabilités ont la même valeur. Le churn naturel domine la dynamique du système, par conséquent, le temps nécessaire pour que les



(a) $\alpha = \alpha^{(1)}$



(b) $\alpha = \alpha^{(2)}$

FIGURE 7.11 – $R(v, m, l, \ell)$ en fonction du nombre m d'événements `join` et `leave` dans le système dans le cas où $v = 2048$, $\mu = 0.25$, $\gamma = 7$, $\Gamma = 14$

TABLE 7.1 – Gain d'un `lookup` en fonction du nombre de chemins indépendants employé dans le cas où $\upsilon = 2048, m = 0, \ell = 11$ et $\alpha = \alpha^{(2)}$.

l	1	2	3	4	5	6	7	8	9	10	11
R	0.386	0.599	0.717	0.783	0.819	0.839	0.85	0.856	0.86	0.861	0.862
G	0.0	55.2	85.8	103.	112.	117.	120.	122.	123.	123.	123.
G_l	*	55.2	19.7	9.21	4.60	2.44	1.31	0.71	0.47	0.12	0.12

clusters effectuent une opération `split` ou `merge` est inférieur au temps de pollution d'un cluster. Dans ce cas, le routage redondant sur l chemins n'apporte aucun gain.

À l'inverse, lorsque la distribution initiale des clusters comporte des états pollués ($\alpha = \alpha^{(2)}$), le système comporte des clusters pollués, impactant ainsi les probabilités de `lookup` simple $L(\upsilon, m, \ell)$ et redondant $R(\upsilon, m, l, \ell)$. La figure 7.11(b) illustre l'impact de cette distribution initiale sur les valeurs de ces probabilités. Dans le cas où $l = 1$, on n'utilise que le `lookup` simple et moins de 40% des requêtes de `lookup` atteignent leur destination. À l'inverse, on peut constater que dès qu'on utilise $l = 4$ chemins indépendants parmi les 12 existants, cette probabilité de succès avoisine les 80%.

Le tableau 7.1 fournit les valeurs de $R(\upsilon, m, l, \ell)$ dans le système à $\upsilon = 2048$ clusters à l'instant initial $m = 0$ pour différents taux de redondances $1 \leq l \leq \ell$. La seconde ligne du tableau illustre le gain absolu G obtenu entre $R(\upsilon, m, l, \ell)$ et un `lookup` simple. Ce gain est calculé ainsi :

$$G = 100 \times \frac{R(\upsilon, m, l, \ell) - R(\upsilon, m, 1, \ell)}{R(\upsilon, m, 1, \ell)}.$$

Enfin, la troisième ligne fournit le gain relatif :

$$\forall 2 \leq l \leq \ell, G_l = 100 \times \frac{R(\upsilon, m, l, \ell) - RR(\upsilon, m, l-1, \ell)}{R(\upsilon, m, l-1, \ell)}.$$

Le gain relatif transcrit l'augmentation du taux de succès des requêtes de `lookup` lorsqu'on incrémente le nombre l de chemins indépendants utilisés. Par conséquent, le gain relatif n'est défini que pour les valeurs de $2 \leq l \leq \ell$. On constate alors qu'avec $l = 4$ on double le taux de succès par rapport à un `lookup` classique. Le gain absolu se stabilise ensuite et le gain relatif tend alors vers zéro. Ceci illustre le compromis à trouver entre le taux de succès souhaité et le coût engendré en termes de messages dans l'overlay.

7.3 Conclusion

Nous avons présenté dans ce chapitre une évaluation de l'ensemble de la DHT PeerCube. Nous avons modélisé le comportement des clusters de PeerCube au moyen d'une chaîne de Markov à temps discret. Nous avons étudié la proportion de clusters sains et pollués dans le système. Malgré un grand nombre d'entités malveillantes, celles-ci sont incapables de polluer l'intégralité du système dès que des identifiants à validité limitée sont mis en place.

Cependant ces identifiants à validité limitée induisent un churn supplémentaire dans la DHT. Nous avons étudié l'impact des identifiants à validité limitée sur la dynamique du système. Nous avons montré que l'impact de cette technique reste négligeable vis-à-vis des du churn naturel dans le système.

Enfin, nous avons étudié l'impact de la présence d'entités malveillantes dans le système sur les opérations de recherche `lookup` (et donc les opérations `put` et `get`). Bien que ces entités ne puissent pas polluer l'intégralité du système, leur présence peut nuire à ces opérations. Ainsi, une requête de `lookup` classique n'atteint sa destination que dans 40% des cas dans un système comportant des entités malveillantes à l'instant initial. PeerCube permet à ces requêtes d'emprunter des chemins indépendants. On a montré dans ce chapitre que cette manière permet de tolérer la présence de clusters pollués sans pour autant employer tous les chemins indépendants.

Bibliographie

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

[ACLS13] E. Anceaume, F. Castella, R. Ludinard et B. Sericola : Markov Chains Competing for Transitions : Application to Large-Scale Distributed Systems. *Methodology and Computing in Applied Probability*, 15(2):305–332, juin 2013.

[ALS12] E. Anceaume, R. Ludinard et B. Sericola : Performance evaluation of large-scale dynamic systems. *ACM SIGMETRICS Performance Evaluation Review*, 39(4):108–117, avril 2012.

[ASLT11] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel : Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, juin 2011.

[SL04] M. Srivatsa et L. Liu : Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems : A quantitiative Analysis. Dans *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC, 2004.

[SS88] Y. Saad et M. Schultz : Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7), 1988.

Chapitre 8

Conclusion

Problématique, enjeux et solutions

Nous nous sommes intéressés dans le cadre de cette thèse aux systèmes large échelle et à la caractérisation de fautes au sein de ceux-ci. Les systèmes large échelle sont des systèmes composés d'un très grand nombre d'entités. Ces entités sont connectées entre elles au travers du réseau afin de rendre ou consommer un ou des services.

Ces entités ainsi que les éléments du réseau sont sujets à des fautes aux conséquences variables. Ces fautes sont inévitables. Dès lors, il est naturel dans le cas du déploiement d'un tel système de souhaiter détecter l'occurrence des fautes aussi rapidement que possible après leur activation.

La supervision consiste à collecter et analyser diverses données du système considéré afin de connaître son état à tout instant. Le nombre d'entités et d'éléments réseau impliqués dans un tel système rend la supervision de chacun d'entre eux inenvisageable. Afin de pallier ce problème, nous utilisons ici des mesures de qualité relatives aux services consommés par les utilisateurs du système. Une dégradation de qualité est une défaillance. Celle-ci est significative de l'activation d'une faute dans le système ou le réseau d'interconnexion.

Une faute peut avoir un impact variable. Dans certains cas, seul un faible nombre d'entités perçoit une dégradation de qualité tandis que dans d'autres cas, un grand nombre d'entités perçoivent une défaillance. La rédaction de ce document a été guidée par la question suivante : dans quelle mesure et de quelle manière est-il possible du point de vue d'un utilisateur d'un système large échelle de distinguer les fautes massives des fautes isolées qui apparaissent au sein de celui-ci ?

En nous basant sur les variations de qualités perçues par les entités et l'hypothèse que des défaillances similaires sont la conséquence de l'activation d'une même faute, nous avons mis en place un modèle simple permettant de construire des scénarios expliquant ces défaillances.

Caractérisation de fautes Ce modèle repose sur une hypothèse de partitionnement. Entre deux observations effectuées aux instants $k-1$ et k, on considère qu'une entité

perçoit au plus une défaillance. De plus, on considère qu'une faute a le même impact sur toutes les entités percevant la défaillance qui en résulte. Enfin, on ne considère que les scénarios dans lesquels des entités percevant des variations de qualité similaires sont dues à un nombre restreint de fautes. Ce modèle est basée sur deux paramètres : le rayon de cohérence r et le seuil de densité τ. Nous faisons l'hypothèse dans le modèle décrit au chapitre 3 qu'une faute a un impact similaire sur toutes les entités supervisées qui la perçoivent. En particulier, si un ensemble B d'entités supervisées forme un ensemble r-cohérent avant la perception de la défaillance (*i.e.*, à l'instant $k-1$), alors celles-ci forment encore un ensemble r-cohérent à l'instant k et donc par la définition 4, cet ensemble a un mouvement r-cohérent dans l'intervalle de temps $[k-1, k]$. Malgré ce modèle simple, nous avons montré dans le chapitre 3 qu'il est impossible de déterminer de manière certaine pour chaque entité percevant une défaillance, si celle-ci est due à une faute isolée ou à une faute massive.

Cependant, il est possible de classer les entités percevant une défaillance en trois sous-ensembles disjoints. La première classe correspond aux entités dont il est certain qu'elles ont perçu une défaillance due à une faute isolée, le second aux entités dont il est certain qu'elles ont perçu une défaillance due à une faute massive. Le dernier ensemble contient l'ensemble des entités pour lesquelles il existe une incertitude sur le type de faute ayant induit la défaillance perçue.

Afin de construire ces ensembles, nous plaçons les entités supervisées dans un espace des qualités. Pour chacun de ces sous-ensembles, nous avons fourni des conditions nécessaires et suffisantes portant sur le voisinage de chaque entité dans cet espace des qualités. De plus, nous avons défini une relation d'équivalence entre les entités, nous permettant ainsi de réduire le nombre total de calculs pour la classification des entités dans chacun des trois ensembles considérés. L'utilisation de l'espace des qualités ainsi que la notion de similarité employée sont au centre de ce modèle. En particulier, deux entités dont les trajectoires dans l'espace des qualités sont trop éloignées sont considérées comme ayant perçu des défaillances dues à des fautes différentes. Par conséquent, cette hypothèse nous permet de paralléliser la caractérisation de fautes entre les entités supervisées. De plus, cette approche répartie entre les entités supervisées produit le même résultat que celui que calculerait un observateur global du système.

Mise en œuvre algorithmique Nous avons présenté dans le chapitre 4 les algorithmes nécessaires à la mise en œuvre de cette caractérisation de manière distribuée. l'approche répartie est moins coûteuse que l'approche centralisée car elle travaille sur des ensembles d'entités de plus petite taille. Néanmoins, dans le cas où toutes les entités perçoivent la même variation de qualité, le coût de l'approche répartie est le même que celui de l'approche centralisée.

Le coût des différents algorithmes est évalué par simulation. Nous avons ainsi montré que l'approche heuristique employant le corollaire 2 se montre nettement moins coûteuse que la solution optimale employant le théorème 4 avec une caractérisation parfaite dans quasiment 100% des cas concernés. D'autre part, ces simulations ont permis de montrer l'importance de la fréquence d'échantillonnage des états du système sur la qualité de la caractérisation produite. En effet, une fréquence accrue permet de réduire

la proportion d'entités en état indéterminé. Enfin, ces simulations ont permis de montrer que les hypothèses sur lesquelles s'appuient la construction du modèle ne sont pas trop restrictives. En effet, tant que la fréquence d'échantillonnage des états du système n'est pas trop faible, le nombre de mauvaises caractérisations est négligeable.

FixMe La caractérisation des fautes que nous avons proposé dans ce document est basée sur les qualités de service perçues par les entités supervisées. Nous avons décrit dans le chapitre 5 FixMe, une architecture auto-organisante dans laquelle les entités du système se placent en fonction des qualités de service qu'elles perçoivent.

Cette architecture repose sur une décomposition de l'espace des qualités en parties élémentaires et indivisibles appelées buckets. Les buckets sont regroupés en cellules en fonction de la distribution des entités dans l'espace des qualités. La population de chaque cellule est organisée au sein d'une table de hachage distribuée nommée PeerCube. À chaque instant discret k, les entités évaluent la qualité de chaque service qu'elles consomment et se replace dans FixMe si celles-ci ont varié depuis l'instant précédent $k-1$. Si cette variation est considérée comme anormale, l'entité écrit dans la table de hachage distribuée sous-jacente. De cette manière, toutes les entités ayant perçu une variation similaire de qualité sont capables de retrouver les autres entités et ainsi mettre en œuvre la caractérisation présentée au chapitre 3.

La structure de FixMe crée naturellement un compromis entre le nombre de cellules dans l'espace des qualités et le nombre d'entités dans ces cellules. La complexité algorithmique des opérations de FixMe est analysée sous deux distributions des entités dans l'espace des qualités.. La première place toutes les entités au sein d'un même bucket et donc dans une même instance PeerCube, tandis que la seconde maximise le nombre de cellules et réduit le nombre moyen d'entités dans chaque instance PeerCube. Dans les deux cas, les opérations de `join`, `leave` et `lookup` nécessitent un nombre logarithmique de messages. Les opérations `split` et `merge` ne dépendent quant à elles que du partitionnement de l'espace des qualités.

Évaluation de PeerCube Les chapitres 6 et 7 sont dédiés à une évaluation de performance de PeerCube. Le chapitre 6 décrit le fonctionnement de cette table de hachage distribuée, ainsi que le comportement d'un adversaire souhaitant manipuler celle-ci. Nous avons montré dans ce chapitre que l'utilisation d'identifiants à durée de validité permettait de limiter l'action de cette adversaire au niveau élémentaire de PeerCube. Nous avons de plus étudié l'évolution de la composition d'un cluster en présence d'un adversaire en fonction des entrées et des sorties des entités de celui-ci.

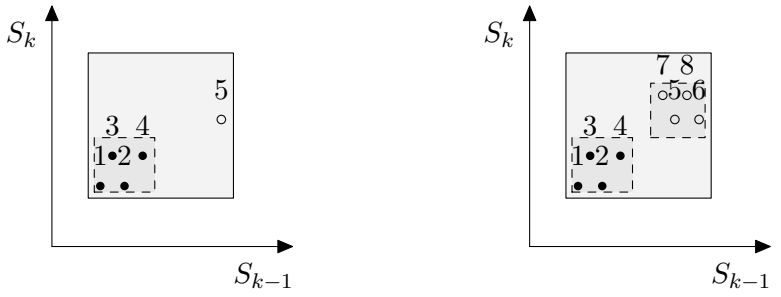
Dans le chapitre 7 nous avons utilisé la modélisation introduite au chapitre précédent afin d'étudier la dynamique de PeerCube dans son ensemble. Nous nous sommes plus particulièrement focalisé sur deux aspects : le nombre de clusters sains et pollués dans le système et les changements topologiques au sein de celui-ci. Nous avons ainsi montré la résilience de PeerCube au churn des entités ainsi qu'aux comportements malveillants.

Perspectives

Évaluation en présence de densité variable La caractérisation présentée au chapitre 3 est basée sur deux paramètres : le rayon de cohérence r et le seuil de densité τ. Ces deux paramètres permettent de définir une densité qui est employée dans notre caractérisation. Par conséquent, une surestimation des paramètres τ et r peut nous amener à regrouper ensemble des entités ayant perçu des défaillances dues à des fautes différentes. D'autre part, dans le modèle que nous avons proposé dans ce document, ces paramètres sont constants et indépendants des qualités perçues par les entités. Or, il semble raisonnable d'imaginer que le rayon de cohérence d'une défaillance augmente lorsque la qualité perçue par les entités diminue. Par exemple, dans le cas d'une défaillance liée à un équipement réseau, la variance des qualités des entités impactées est plus faible dans le cas d'entités percevant de bonnes qualités (en ville par exemple) que dans le cas d'entités habituées à des qualités moyennes (en campagne par exemple). Nous travaillons actuellement sur une génération de vérités synthétiques plus générale que celle employée dans le chapitre 4. Cette nouvelle approche nous permet ainsi de générer des mouvements de densités différentes et de générer un nombre variable de fautes entre deux instants discrets $k-1$ et k. Ces travaux nous permettront de mesurer l'impact d'un mauvais paramétrage du rayon de cohérence r et du seuil de densité τ sur la caractérisation proposée.

Tolérer des densités variables Comme nous l'avons vu, malgré le modèle simple que nous avons considéré, il est impossible d'attribuer à chaque entité percevant une défaillance le type de faute qui l'a causée. Nous avons donc affaibli le problème considéré en introduisant la notion d'état indéterminé. Les états indéterminés sont dus à la superposition de plusieurs anomalies comme illustré dans chapitre 4. Comme nous pouvons le voir sur la figure 8.1, toutes les superpositions d'anomalies n'engendrent pas des états indéterminés. En effet, la figure 8.1(a) illustre la superposition d'une faute massive impactant les entités $1, 2, 3$ et 4 et d'une faute isolée impactant l'entité 5. À l'inverse, la figure 8.1(b), illustre la superposition de deux fautes massives impactant respectivement les entités $1, 2, 3, 4$ et $5, 6, 7, 8$. De la même manière, l'algorithme DBSCAN [EKSX96] que nous avons présenté dans la section 4.3 ne gère pas les clusters de densités différentes. Ce problème est résolu dans OPTICS [ABKS99] en introduisant une notion de distance différente. Il semble donc intéressant de suivre la même piste pour ce modèle afin de raffiner la caractérisation de fautes massives et de limiter les erreurs de caractérisation. Ainsi, tolérer des densités variables au sein de cette caractérisation afin d'adapter le rayon de cohérence et le seuil de densité en fonction de la position considérée dans l'espace des qualités permettrait d'affiner cette caractérisation en fonction de la position dans l'espace des qualités et ainsi réduire les erreurs de caractérisation.

Mesures de vraisemblance Dans le cas où la superposition d'anomalies engendre des états indéterminés, il semble intéressant de pouvoir associer à chaque entité en état indéterminé une mesure de vraisemblance. En effet, comme on peut le voir sur la



(a) 1 faute isolée et 1 faute massive

(b) 2 fautes massives

FIGURE 8.1 – Superpositions d'anomalies ne menant pas à des états indéterminés.

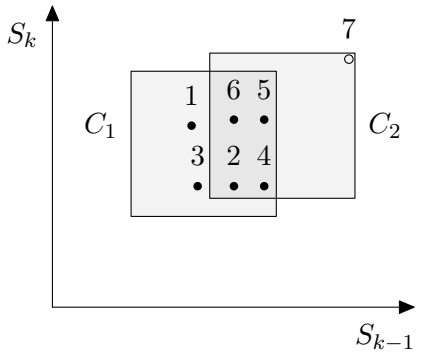


FIGURE 8.2 – Superposition d'anomalies massives menant à des états indéterminés.

figure 8.2, il semble raisonnable de penser que les entités $1, 2, 3, 4, 5, 6$ ont perçu une défaillance due à une même faute massive tandis que l'entité 7 a perçu une défaillance due à une faute isolée. Dans cet exemple, l'entité 7 est la plus distance de chacune des entités de l'ensemble $\{1, 2, 3, 4, 5, 6\}$. Cette intuition est décrite dans [BKNS00]. Il serait donc intéressant d'appliquer ce principe à la caractérisation proposée ici afin de fournir une mesure de vraisemblance sur les scenarios possibles. Ainsi dans cet exemple, les entités $1, 3, 7$ sont en état indéterminé. L'application de ce principe permettrait d'attribuer aux entités 1 et 3, une forte probabilité d'avoir perçu une défaillance due à une faute massive et à l'entité 7 une forte probabilité d'avoir perçu une défaillance due à une faute isolée. L'introduction de mesures de vraisemblance permettrait d'apporter un support lors de la recherche des causes de la défaillance perçue et d'orienter celle-ci.

Tolérer la présence d'un adversaire Nous n'avons considéré dans les chapitres 3 à 5 que le cas des fautes non byzantines, c'est-à-dire que les entités suivent le protocole défini et que les défaillances résultant de l'activation de fautes dans le système sont perçues de manière cohérente par les entités. Généraliser l'architecture de FixMe afin

de prendre en compte ce type de faute est une piste d'amélioration de ces travaux. En particulier, l'enjeu majeur d'une telle généralisation porte sur la possibilité de mise en œuvre en présence d'un adversaire souhaitant mettre en défaut la caractérisation proposée. Un tel adversaire peut chercher à masquer les fautes isolées ou à l'inverse générer un grand nombre de prétendues fautes isolées. Ainsi, en déplaçant des entités malveillantes au sein de FixMe, un adversaire est en mesure de mettre en défaut la caractérisation proposée dans ce document. Dans quelle mesure est-il possible de généraliser l'architecture de FixMe afin de tolérer un tel comportement, ou, à défaut, de limiter les conséquences de celui-ci ?

Validation expérimentale D'autre part, une implémentation en vue de tests réels permettrait de valider définitivement ce modèle ainsi que la caractérisation proposée. Deux difficultés majeures sont à prendre en compte dans le cadre d'une telle validation. Cela nécessite tout d'abord de déployer un prototype sur un nombre suffisamment important d'entités et d'autre part d'être capable de comparer les résultats obtenus à une vérité terrain.

Application à d'autres domaines Enfin, il semble intéressant d'appliquer cette caractérisation à d'autres champ d'applications. De récent travaux [GCR09, SH11, SC13] dans le domaine de la vision par ordinateur portent sur la détection de groupe au sein d'une foule. Il serait donc intéressant de comparer notre approche à celles-ci dans le contexte de la supervision de foules. En effet, dans ce contexte, la caractérisation décrite dans ce document permettrait d'identifier d'une part les individus se déplaçant en groupe au sein de la foule, et d'autre part ceux ayant des mouvements individuels au sein de celle-ci. Cette identification pourrait par exemple être utile dans l'analyse de vidéos de surveillance.

Bibliographie

[ABKS99] M. Ankerst, M. M. Breunig, H. P. Kriegel et J. Sander : OPTICS : Ordering Points To Identify the Clustering Structure. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[ABL+14a] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola : Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, juin 2014.

[ABL+14b] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand, B. Sericola et G. Straub : Anomaly Characterization Problems. Dans *16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, 2014.

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans

Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO, pages 15–24, 2008.

[ACLS13] E. Anceaume, F. Castella, R. Ludinard et B. Sericola : Markov Chains Competing for Transitions : Application to Large-Scale Distributed Systems. *Methodology and Computing in Applied Probability*, 15(2):305–332, juin 2013.

[ALL+12] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : A Self-organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. Dans *Proceedings of the 16th International Conference On Principles Of Distributed Systems*, OPODIS, pages 1–12, décembre 2012.

[ALL+13] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : Détection Répartie de Défaillances Isolées. Dans *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, mai 2013.

[ALS12] E. Anceaume, R. Ludinard et B. Sericola : Performance evaluation of large-scale dynamic systems. *ACM SIGMETRICS Performance Evaluation Review*, 39(4):108–117, avril 2012.

[ASLT11] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel : Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, juin 2011.

[BKNS00] M. M. Breunig, H. P. Kriegel, R. T. Ng et J. Sander : LOF : Identifying Density-based Local Outliers. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[EKSX96] M. Ester, H. P. Kriegel, J. Sander et X. Xu : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Dans *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 226–231, 1996.

[GCR09] W. Ge, R. T. Collins et B. Ruback : Automatically detecting the small group structure of a crowd. Dans *IEEE Workshop on Applications of Computer Vision*, WACV, pages 1–8, 2009.

[SC13] F. Solera et S. Calderara : Social Groups Detection in Crowd through Shape-Augmented Structured Learning. Dans *Image Analysis and Processing – ICIAP 2013*, volume 8156, pages 542–551, 2013.

[SH11] J. Sochman et D. C. Hogg : Who knows who - Inverting the Social Force Model for finding groups. Dans *Proceedings of the IEEE International Conference on Computer Vision Workshops*, ICCV, pages 830–837, november 2011.

Liste des publications

Revues internationales avec comité de lecture

[1] A-M. Kermarrec, E. Le Merrer, N. Le Scouarnec, R. Ludinard, P. Maillé, G. Straub et A. Van Kempen. Performance evaluation of a peer-to-peer backup system using buffering at the edge. Dans *Computer Communications*, Octobre 2014.

[2] R. Ludinard, E. Totel, F. Tronel, V. Nicomette, M. Kaaniche, E. Alata, R. Akrout et Y. Bachy. An Invariant-based Approach for Detecting Attacks against Data in Web Applications. Dans *International Journal of Secure Software Engineering (IJSSE)*, Janvier 2014.

[3] E. Anceaume, F. Castella, R. Ludinard et B. Sericola. Markov Chains Competing for Transitions : Application to Large-Scale Distributed Systems. Dans *Methodology and Computing in Applied Probability*, 15(2) :305–332, Juin 2013.

[4] E. Anceaume, R. Ludinard et B. Sericola. Performance evaluation of large-scale dynamic systems. Dans *ACM SIGMETRICS Performance Evaluation Review*, 39(4) :108–117, Avril 2012.

[5] E. Anceaume, F. Brasileiro, R. Ludinard, B. Sericola, and F. Tronel. Dependability Evaluation of Cluster-based Distributed Systems. Dans *International Journal of Foundations of Computer Science*, 22(5) :1123–1142, Août 2011.

Conférences internationales avec comité de lecture

[1] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola. Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, Juin 2014.

[2] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola, et G. Straub. FixMe : A Self-Organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. Dans *Proceedings of the 16th International Conference On Principles Of Distributed Systems*, OPODIS, pages 1–15, Decembre 2012.

[3] R. Ludinard, E. Totel, F. Tronel, V. Nicomette, M. Kaaniche, E. Alata, R. Akrout et Y. Bachy. Detecting Attacks Against Data in Web Applications. Dans *Proceedings of the 7th International Conference on Risks and Security of Internet and Systems*, CRiSIS, pages 1–8, Octobre 2012.

[4] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel. Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, Juin 2011.

[5] E. Anceaume, R. Ludinard et B. Sericola. Analytic Study of the Impact of Churn in Cluster-Based Structured P2P Overlays. Dans *Proceedings of the Internation Conference on Communications*, ICC, pages 302–307, Mai 2010.

[6] E. Anceaume, F. Brasiliero, R. Ludinard, B. Sericola et F. Tronel. Brief Announcement : Induced Churn to Face Adversarial Behavior in Peer-to-Peer Systems. In *Proceedings of the 11th International Symposium on Stabilization, Safety et Security of Distributed Systems*, SSS, pages 773–774, Novembre 2009.

[7] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja. Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

Workshops et conférences francophones

[1] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand, B. Sericola, and G. Straub. Anomaly characterization problems. Dans *16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, 2014.

[2] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub. A Self-organising Isolated Anomaly Detection Architecture for Large Scale Systems. Dans *Nem-Summit*, Octobre 2013.

[3] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub. FixMe : détection répartie de défaillances isolées. Dans *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, Mai 2013.

[4] R. Ludinard, L. Le Hennaff et E. Totel. RRABIDS, un système de détection d'intrusion pour les applications Ruby on Rails. Dans *Actes du Symposium 2011 sur la Sécurité des Technologies de l'Information et des Communications*, SSTIC, Juin 2011.

[5] E. Anceaume, R. Ludinard, B. Sericola et F. Tronel. Modélisation et Évaluation des Attaques Ciblées dans un Overlay Structuré. Dans *Colloque Francophone sur l Ingénierie des Protocoles*, CFIP, 2011.

[6] E. Anceaume, R. Ludinard, F. Tronel, F. Brasiliero, and B. Sericola. Analytical Study of Adversarial Strategies in Cluster-based Overlays. Dans *2nd International Workshop on Reliability, Availability et Security*, WRAS, pages 293–298, 2009.

Brevets

[1] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub. Method for isolated anomaly detection in large-scale audio/video/data processing systems. patent no. 13306029.3, Juillet 2013.

[2] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub. Method for isolated anomaly detection in large-scale data processing systems. patent no. 12306237.4, Octobre 2012.

Références

[ABKS99] M. Ankerst, M. M. Breunig, H. P. Kriegel et J. Sander : OPTICS : Ordering Points To Identify the Clustering Structure. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[ABL+09] E. Anceaume, F. Brasiliero, R. Ludinard, B. Sericola et F. Tronel : Brief Announcement : Induced Churn to Face Adversarial Behavior in Peer-to-Peer Systems. Dans *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS, pages 773–774, novembre 2009.

[ABL+11] E. Anceaume, F. Brasiliero, R. Ludinard, B. Sericola et F. Tronel : Dependability Evaluation of Cluster-based Distributed Systems. *International Journal of Foundations of Computer Science*, 22(5):1123–1142, août 2011.

[ABL+14a] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand et B. Sericola : Anomaly Characterization in Large Scale Networks. Dans *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN, juin 2014.

[ABL+14b] E. Anceaume, Y. Busnel, E. Le Merrer, R. Ludinard, J-L. Marchand, B. Sericola et G. Straub : Anomaly Characterization Problems. Dans *16èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, 2014.

[ABLR08] E. Anceaume, F. Brasileiro, R. Ludinard et A. Ravoaja : Peercube : A hypercube-based p2p overlay robust against collusion and churn. Dans *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO, pages 15–24, 2008.

[ACLS13] E. Anceaume, F. Castella, R. Ludinard et B. Sericola : Markov Chains Competing for Transitions : Application to Large-Scale Distributed Systems. *Methodology and Computing in Applied Probability*, 15(2):305–332, juin 2013.

[ALL+12] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : A Self-organizing Isolated Anomaly Detection Architecture for Large Scale Distributed Systems. Dans *Proceedings of the 16th*

International Conference On Principles Of Distributed Systems, OPODIS, pages 1–12, décembre 2012.

[ALL+13a] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : A Self-organising Isolated Anomaly Detection Architecture for Large Scale Systems. Dans *Nem-Summit*, octobre 2013.

[ALL+13b] E. Anceaume, E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : FixMe : Détection Répartie de Défaillances Isolées. Dans *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, AlgoTel, pages 1–4, mai 2013.

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell et C. Landwehr : Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, janvier 2004.

[ALS10] E. Anceaume, R. Ludinard et B. Sericola : Analytic Study of the Impact of Churn in Cluster-Based Structured P2P Overlays. Dans *Proceedings of the Internation Conference on Communications*, ICC, pages 302–307, mai 2010.

[ALS12] E. Anceaume, R. Ludinard et B. Sericola : Performance evaluation of large-scale dynamic systems. *ACM SIGMETRICS Performance Evaluation Review*, 39(4):108–117, avril 2012.

[ALST11] E. Anceaume, R. Ludinard, B. Sericola et F. Tronel : Modélisation et Évaluation des Attaques Ciblées dans un Overlay Structuré. Dans *Colloque Francophone sur l'Ingénierie des Protocoles*, CFIP, 2011.

[ALT+09] E. Anceaume, R. Ludinard, F. Tronel, F. Brasiliero et B. Sericola : Analytical Study of Adversarial Strategies in Cluster-based Overlays. Dans *Proceedings of the 2nd International Workshop on Reliability, Availability, and Security*, WRAS, pages 293–298, 2009.

[ARC13] ARCEP : Haut et très haut débit sur réseaux fixes au 30 septembre 2013. `http://www.arcep.fr/index.php?id=12115`, novembre 2013.

[AS07] B. Awerbuch et C. Scheideler : Towards Scalable and Robust Overay Networks. Dans *Proceedings of the International Workshop on Peer-to-Peer Systems*, IPTPS, 2007.

[ASLT11] E. Anceaume, B. Sericola, R. Ludinard et F. Tronel : Modeling and Evaluating Targeted Attacks in Large Scale Dynamic Systems. Dans *Proceedings of the 41rst International Conference on Dependable Systems and Networks*, DSN, pages 347–358, juin 2011.

[BBG08] K. Buchin, M. Buchin et J. Gudmundsson : Detecting Single File Movement. Dans *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2008.

[BEF84] J. C Bezdek, R. Ehrlich et W. Full : FCM : The Fuzzy C-Means Clustering Algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.

[BGHW06] M. Benkert, J. Gudmundsson, F. Hübner et T. Wolle : Reporting flock patterns. Dans *Proceedings of the 14th European Symposium on Algorithms*, ESA, pages 660–671, 2006.

[BGHW08] M. Benkert, J. Gudmundsson, F. Hübner et T. Wolle : Reporting Flock Patterns. *Computational Geometry Theory and Applications*, 41(3): 111–125, novembre 2008.

[BKNS00] M. M. Breunig, H. P. Kriegel, R. T. Ng et J. Sander : LOF : Identifying Density-based Local Outliers. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider et B. Seeger : The R*-tree : An Efficient and Robust Access Method for Points and Rectangles. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[Bro] Broadband Forum : TR-069 CPE WAN Management Protocol Issue 1, Amend.4, 2011.

[CBG10] D. R. Choffnes, F. E. Bustamante et Z. Ge : Crowdsourcing Service-level Network Event Monitoring. Dans *Proceedings of the ACM SIGCOMM Conference*, pages 387–398, 2010.

[CFNV04] M. Correia, N. Ferreira Neves et P. Veríssimo : How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems. Dans *Proceedings of the 23rd International Symposium on Reliable Distributed Systems*, SRDS, pages 174–183, 2004.

[CFSD90] J. D. Case, M. Fedor, M. L. Schoffstall et J. Davin : Simple Network Management Protocol (SNMP). Rapport technique, IETF, 1990.

[Cho10] D. R. Choffnes : *Service-Level Network Event Detection from Edge Systems*. Thèse de doctorat, Northeastern University, 2010.

[Def77] D. Defays : An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[DLR77] A. P. Dempster, N. M. Laird et D. B. Rubin : Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society : Series B*, 39:1–38, 1977.

[EJ01] D. Eastlake et P. Jones : US Secure Hash Algorithm 1 (SHA1). Rapport technique, IETF, 2001.

[EKSX96] M. Ester, H. P. Kriegel, J. Sander et X. Xu : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Dans *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 226–231, 1996.

[FSY05] A. Fiat, J. Saia et M. Young : Making Chord Robust to Byzantine Attacks. Dans *Proceedings of the 13rd European Symposium on Algorithms*, ESA, pages 803–814, 2005.

[GCR09] W. Ge, R. T. Collins et B. Ruback : Automatically detecting the small group structure of a crowd. Dans *IEEE Workshop on Applications of Computer Vision*, WACV, pages 1–8, 2009.

[HCDW12] C. Y. Hong, M. Caesar, N. Duffield et J. Wang : Tiresias : Online Anomaly Detection for Hierarchical Operational Network Data. Dans *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems*, ICDCS, pages 173–182, 2012.

[HE02] G. Hamerly et C. Elkan : Alternatives to the K-means Algorithm That Find Better Clusterings. Dans *Proceedings of the 11th International Conference on Information and Knowledge Management*, CIKM, pages 600–607, 2002.

[HFPS99] R. Housley, W. Ford, W. Polk et D. Solo : Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile. Rapport technique, IETF, 1999.

[Hol04] C. C. Holt : Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[HW79] J. A. Hartigan et M. A. Wong : Algorithm AS 136 : A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[JLO07] C. S. Jensen, D. Lin et Beng-Chin O. : Continuous Clustering of Moving Objects. *Knowledge and Data Engineering, IEEE Transactions on*, 19(9): 1161–1174, 2007.

[JYZ+08] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen et H. T. Shen : Discovery of Convoys in Trajectory Databases. *Proceedings VLDB Endowment*, 1(1): 1068–1080, août 2008.

[KAD+07] R. Kotla, L. Alvisi, M. Dahlin, A. Clement et E. Wong : Zyzzyva : Speculative Byzantine Fault Tolerance. Dans *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP, pages 45–58, 2007.

[Kal60] R. E. Kalman : A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45, 1960.

[KLL+ar] A-M. Kermarrec, E. Le Merrer, N. Le Scouarnec, R. Ludinard, P. Maillé, G. Straub et A. Van Kempen : Performance evaluation of a peer-to-peer backup system using buffering at the edge. *Computer Communications*, *to appear*.

[KR87] L. Kaufman et P. Rousseeuw : *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. 1987.

[Lap96] J. C. Laprie : *Guide de la sûreté de fonctionnement*. Cépaduès-Editions, 1996.

[LLHT11] R. Ludinard, Loic Le Hennaff et E. Totel : RRABIDS, un système de détection d'intrusion pour les applications Ruby on Rails. Dans *Actes du Symposium 2011 sur la Sécurité des Technologies de l'Information et des Communications*, SSTIC, juin 2011.

[LLSS12] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : Method for Isolated Anomaly Detection in Large-scale Data Processing Systems. patent no. 12306237.4, octobre 2012.

[LLSS13] E. Le Merrer, R. Ludinard, B. Sericola et G. Straub : Method for Isolated Anomaly Detection in Large-scale Audio/Video/Data Processing Systems. patent no. 13306029.3, juillet 2013.

[LSW06] T. Locher, S. Schmid et R. Wattenhofer : eQuus : A Provably Robust and Locality-Aware Peer-to-Peer System. Dans *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, P2P, pages 3–11, 2006.

[LTT+12] R. Ludinard, E. Totel, F. Tronel, V. Nicomette, M. Kaaniche, E. Alata, R. Akrout et Y. Bachy : Detecting Attacks Against Data in Web Applications. Dans *Proceedings of the 7th International Conference on Risks and Security of Internet and Systems*, CRiSIS, pages 1–8, octobre 2012.

[LTT+ar] R. Ludinard, E. Totel, F. Tronel, V. Nicomette, M. Kaaniche, E. Alata, R. Akrout et Y. Bachy : An Invariant-based Approach for Detecting Attacks against Data in Web Applications. *International Journal of Secure Software Engineering (IJSSE)*, *to appear*.

[MA04] S. Malgorzata et S. S. Adarshpal : A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[MCC03] M. L. Massie, B. N. Chun et D. E. Culler : The Ganglia Distributed Monitoring System : Design, Implementation And Experience. *Parallel Computing*, 30:2004, 2003.

[MM02] P. Maymounkov et D. Mazières : Kademlia : A Peer-to-Peer Information System Based on the XOR Metric. Dans *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS, pages 53–65, 2002.

[MMBK10] D. Mills, J. Martin, J. Burbank et W. Kasch : Network Time Protocol Version 4 : Protocol and Algorithms Specification. Rapport technique, IETF, 2010.

[Pag54] E. S. Page : Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, juin 1954.

[PM00] D. Pelleg et A. Moore : X-means : Extending K-means with Efficient Estimation of the Number of Clusters. Dans *Proceedings of the 17th International Conference on Machine Learning*, pages 727–734, 2000.

[PP06] K. Park et V. S. Pai : CoMon : A Mostly-scalable Monitoring System for PlanetLab. *SIGOPS Operating Systems Review*, 40(1):65–74, janvier 2006.

[RD01] A. Rowstron et P. Druschel : Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Dans *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Middleware, pages 329–350, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp et S. Shenker : A Scalable Content-addressable Network. Dans *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 161–172, 2001.

[Riv92] R. Rivest : The MD5 Message-Digest Algorithm. Rapport technique, IETF, 1992.

[RLH06] Y. Rekhter, T. Li et S. Hares : RFC 4271 : A Border Gateway Protocol 4 (BGP-4). Rapport technique, IETF, 2006.

[RS89] G. Rubino et B. Sericola : Sojourn times in Markov processes. *Journal of Applied Probability*, 26(4):744–756, 1989.

[SC13] F. Solera et S. Calderara : Social Groups Detection in Crowd through Shape-Augmented Structured Learning. Dans *Image Analysis and Processing – ICIAP 2013*, volume 8156, pages 542–551, 2013.

[SEKX98] J. Sander, M. Ester, H.P. Kriegel et X. Xu : Density-Based Clustering in Spatial Databases : The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[Ser90] B. Sericola : Closed form solution for the distribution of the total time spent in a subset of states of a Markov process during a finite observation period. *Journal of Applied Probability*, 27(3):713–719, 1990.

[SH11] J. Sochman et D. C. Hogg : Who knows who - Inverting the Social Force Model for finding groups. Dans *Proceedings of the IEEE International Conference on Computer Vision Workshops*, ICCV, pages 830–837, november 2011.

[Sib73] R. Sibson : SLINK : an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[SL04] M. Srivatsa et L. Liu : Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems : A quantitiative Analysis. Dans *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC, 2004.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek et H. Balakrishnan : Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM Computer Communication Review*, 31(4):149–160, août 2001.

[SS88] Y. Saad et M. Schultz : Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7), 1988.

[VBT09] M. R. Vieira, P. Bakalov et V. J. Tsotras : On-line Discovery of Flock Patterns in Spatio-temporal Data. Dans *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS, pages 286–295, 2009.

[VRBV03] R. Van Renesse, K. P. Birman et W. Vogels : Astrolabe : A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions Computer Systems*, 21(2):164–206, mai 2003.

[Win60] P. R. Winters : Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 6:324–342, 1960.

[YD04] P. Yalagandula et M. Dahlin : A Scalable Distributed Information Management System. Dans *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 379–390, 2004.

[YFG+12] H. Yan, A. Flavel, Z. Ge, A. Gerber, D. Massey, C. Papadopoulos, H. Shah et J. Yates : Argus : End-to-End Service Anomaly Detection and Localization From an ISP's Point of View. Dans *Proceedings of the IEEE INFOCOM Conference*, INFOCOM, pages 2756–2760, 2012.

[ZTG+09] Y. Zhao, Y. Tan, Z. Gong, X. Gu et M. Wamboldt : Self-correlating Predictive Information Tracking for Large-scale Production Systems. Dans *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC, pages 33–42, 2009.

Table des figures

Liste des tableaux

Liste des Algorithmes

Résumé

Internet est un réseau de réseaux permettant la mise en œuvre de divers services consommés par les utilisateurs. Malheureusement, chacun des éléments présents dans le réseau ou impliqués dans ces services peut potentiellement exhiber des défaillances. Une défaillance peut être perçue par un nombre variable d'utilisateurs suivant la localisation dans le système de la source de celle-ci. Cette thèse propose un ensemble de contributions visant à déterminer du point de vue d'un utilisateur percevant une défaillance, si celle-ci est perçue par un faible nombre d'utilisateurs (défaillance isolée) ou à l'inverse par un très grand nombre d'utilisateurs (défaillance massive). Nous formalisons dans un premier temps les défaillances par leur impact sur la perception des services consommés par les utilisateurs. Nous montrons ainsi qu'il est impossible, du point de vue d'un utilisateur, de déterminer de manière certaine si une défaillance perçue est isolée ou massive. Cependant, il possible de déterminer de manière certaine pour chaque utilisateur, s'il a perçu une défaillance isolée, massive, ou s'il est impossible de le déterminer. Cette caractérisation est optimale et totalement parallélisable. Dans un second temps, nous proposons une architecture pour la caractérisation de fautes. Les entités du système s'organisent au sein d'une structure à deux niveaux permettant de regrouper ensemble les entités ayant des perceptions similaires et ainsi mener à bien l'approche proposée. Enfin, une analyse probabiliste de la résistance au dynamisme et aux comportements malveillants du second niveau de cette architecture complète ce document.

Abstract

The Internet is a global system of interconnected computer networks that carries lots of services consumed by users. Unfortunately, each element this system may exhibit failures. A failure can be perceived by a variable range of users, according to the location of the failure source. This thesis proposes a set of contributions that aims at determining from a user perception if a failure is perceived by a few number of users (isolated failure) or in contrast by lots of them (massive failure). We formalize failures with respect to their impact on the services that are consumed by users. We show that it is impossible to determine with certainty if a user perceives a local or a massive failure, from the user point of view. Nevertheless, it is possible to determine for each user whether it perceives a local failure, a massive one or whether it is impossible to determine. This characterization is optimal and can be run in parallel. Then, we propose a self-organizing architecture for fault characterization. Entities of the system organize themselves in a two-layered overlay that allows to gather together entities with similar perception. This gathering allows us to successfully apply our characterization. Finally, a probabilistic evaluation of the resilience to dynamism and malicious behaviors of this architecture is performed.